T H E S E présentée

pour l'obtention

du

DIPLOME de DOCTEUR de 3ème CYCLE

à

L'UNIVERSITE PIERRE ET MARIE CURIE

- Paris 6 -

spécialité : Mathématiques

mention : Informatique

par Monsieur Jérôme CHAILLOUX

Sujet de la thèse :

Le modèle VLISP : description, implémentation et évaluation.

Soutenue le 21 Avril 1980 devant la commission composée de :

Monsieur Bernard ROBINET	Président
Monsieur Patrick GREUSSAY	examinateur
Monsieur Jean François PERROT	examinateur
Monsieur Jean VIGNOLLE	examinateur
Monsieur Joachim LAUBSCH	invité.

Je remercie Monsieur Bernard ROBINET et lui suis très reconnaissant de présider le jury de cette thèse.

Je remercie tout particulièrement Monsieur Patrick GREUSSAY qui, depuis mon arrivée à l'Université de Paris 8-Vincennes, m'a accordé toute son amitié, son aide et ses conseils. Beaucoup d'idées de cette thèse sont issues des fréquentes discussions que nous avons eu ensemble. Je souhaite vivement pouvoir poursuivre cette fructueuse collaboration.

Je remercie particulièrement Monsieur Jean-François PERROT pour son appui et ses conseils.

Je remercie Monsieur Jean VIGNOLLE d'avoir bien voulu s'intéresser à mon travail et participer à ce jury.

Je remercie vivement Monsieur Joachim LAUBSCH d'avoir accepté de venir de Londres pour participer à ce jury.

Cette étude a été réalisée au sein du département d'Informatique de l'Université de Paris 8-Vincennes et a pu être menée à terme grâce au soutien de Monsieur Maurice NIVAT au Laboratoire Informatique Théorique et Programmation 258 du CNRS. J'ai également bénéficié du soutien de Monsieur Gerald BENNETT à l'IRCAM.

La version préliminaire de cette thèse a été relue et commentée par Annette CATTENAT, Gérard NOWAK, Gérard PAUL et Harald WERTZ.

Au terme de cette étude, qu'il me soit permis de remercier toute l'équipe du Département d'Informatique de l'Université de Paris 8-Vincennes, qui a su créer dans des conditions matérielles parfois difficiles, un endroit propice au developpement d'idées nouvelles dans une atmosphère chaleureuse et amicale. En particulier, que Louis AUDOIRE, Annette CATTENAT, Yves DEVILLER, Daniel GOOSSENS, Hervé HUITRIC, Yves LECERF, Gérard PAUL, Monique NAHAS, et Harald WERTZ trouvent ici ma profonde gratitude pour la qualité des nombreuses années passées en leur compagnie.

La réalisation matérielle de cet ouvrage a été faite à l'IRCAM, par l'auteur, au moyen :
- d'un ordinateur DEC PDP10 modèle KI
- d'une imprimante électrostatique VERSATEC connectée à un ordinateur DEC PDP11/40, lui-même connecté au PDP10.

La composition de ce texte a mis en oeuvre de nombreux programmes parmi lesquels on peut citer :

- l'admirable éditeur vidéo E de Arthur Samuel [SAMUEL 77], maintenu à l'IRCAM par Raymond Bara et Jean-Louis Richer.

- le système de composition automatique de textes RF, version considérablement améliorée par Jean-Louis Richer du système RUNOFF de DEC [DEC 75b]

- le programme TRIFON de Patrick Greussay qui réalise l'impression sur VERSATEC à partir du PDP11.

- FNSTAR de Raymond Bara qui permet la fabrication des différentes polices de caractères.

- le paragrapheur et éditeur de références croisées de VLISP-10, de l'auteur, qui a réalisé la composition automatique des textes écrits en VLISP des appendices B, D, et G et des textes écrits en langage machine VCMC2 des appendices C, E et F

Les polices de caractères utilisées ont été construites à partir de celles décrites dans [EARNEST 76], [XGPFONT 74] et [ARCHIBALD 77].

Dans ce texte 3 polices de caractères ont été utilisées :

- la police IRCV25.F11 réalisée par l'auteur

- la police GACI25.F11 réalisée par P. Greussay

- la police MIRV25.F11 réalisée également par l'auteur.

Que tous en soient remerciés.

RESUME

Notre étude présente la réalisation de ***trois systèmes*** VLISP (dialecte du langage LISP) développés à l'université de Paris 8 - Vincennes. Ces systèmes ont été réalisés :

- sur micro-processeur à mots de 8 bits (Intel8080/Zilog80)
- sur mini-ordinateur à mots de 16 bits (PDP11)
- sur gros ordinateur à mots de 36 bits (PDP10)

De ces réalisations est extrait un ***modèle d'implémentation.***

Cette étude propose des solutions aux problèmes de construction et d'évaluation de tels systèmes. Ces problèmes sont :

1) La description exhaustive des implémentations.
   Nous proposons une description fondée sur la ***machine référentielle*** VCMC2.

2) La représentation adéquate des objets et des fonctions VLISP.
   Nous avons associé des ***propriétés naturelles*** aux objets dès leur création et nous avons établi une ***typologie fonctionnelle*** de ces objets.

3) L'efficacité de l'interprète (en place, en temps d'exécution et en compréhension). Notre interprète effectue, pour ses besoins propres, une allocation optimale de la mémoire (allocation mesurée en terme d'appels du module CONS). L'accès direct (ne nécessitant qu'un accès mémoire) aux valeurs des objets de type variable et fonction, et la classification des fonctions par types permettent un ***lancement immédiat*** de toutes les fonctions. La transparence de nos méthodes de description est une conséquence naturelle des deux choix précédents.

4) Le ***pouvoir*** des structures de contrôle.
   Notre modèle d'implémentation généralise les structures de contrôle de VLISP **SELF** et **ESCAPE** en intégrant les nouvelles constructions **EXIT**, **WHERE** et **LETF** et en unifiant totalement leur description et leur implémentation.

5) La souplesse d'utilisation du système.
Nous introduisons le nouveau concept de ***CHRONOLOGIE*** qui permet de créer dynamiquement de nouveaux évaluateurs, permettant les traces méta-circulaires.

Une incarnation de notre modèle est donnée sous la forme de la réalisation d'un système VLISP pour la machine référentielle VCMC2.

TABLE DES MATIERES

CHAPITRE 1

INTRODUCTION

1.1 HISTORIQUE CRITIQUE DE L'IMPLEMENTATION DE LISP.

Le langage LISP [BERKELEY 74, WEISSMAN 67, ALLEN 78] est aujourd'hui le langage le plus utilisé dans les domaines de l'Intelligence Artificielle [WINSTON 77] et de la théorie de la programmation [NIVAT 79]. La particularité majeure de LISP est d'avoir pu évoluer de façon naturelle depuis sa naissance en 1960 [McCARTHY 60a, 60b], à mesure qu'apparaissaient de nouveaux besoins. L'échec patent des rares tentatives de standardisations *{Note 1}* a d'une part permis sa survie et d'autre part conduit à une amélioration constante du langage tant en efficacité à l'interprétation qu'en pouvoir de ses concepts.

Il en résulte aujourd'hui une floraison d'implémentations, bien loin du premier LISP 1 [McCARTHY 60a], ayant chacune des spécifications particulières en fonction de leur utilisation et du moment de leur création.

On distingue ainsi 4 grandes familles d'implémentations :

1) La lignée LISP 1.5 [McCARTHY 62]. Descendante directe de LISP 1 qui était un système interactif (avec le "FLEXOWRITER system"), elle a donné le LISP 1.6 [QUAM 72] très proche de LISP 1.5, mais ayant abandonné la liaison par A-liste, décision dont les conséquences ont été monumentales. Par la suite, l'importance capitale de l'intégration des instruments de mise au point devint de plus en plus évidente, et LISP 1.6 fut augmenté d'un éditeur et d'aides à la mise au point pour donner le U.C.I. LISP [BOBROW 73a].

***{Note 1}* ces standards sont aussi prématurés qu'arbitraires. Prématurés car ils tombent en désuétude aussitôt créés (voir le standard de [HEARN 69] puis [MARTI 79]) et ne tiennent pas compte de l'évolution du langage, arbitraires car ils sont essentiellement définis pour satisfaire un programme particulier (REDUCE 2 dans le cas de HEARN [HEARN 73, 74]).**

2) Les systèmes INTERLISP (anciennement BBN LISP) [TEITELMAN 75] orientés sur une utilisation exclusivement interactive, et qui ont donnés naissance à des sous-systèmes tels que CLISP [TEITELMAN 73] ou DLISP [TEITELMAN 77] qui font une utilisation intensive de périphériques interactifs tels les écrans à haute résolution.

3) MACLISP [MOON 74, LAUBSCH 76] au M.I.T. dont le développement s'est réalisé en symbiose avec celui du système MACSYMA [MACSYMA 75] et qui a inspiré la conception de la machine LISP du M.I.T. [WEINREB 79].

4) Enfin les systèmes VLISP [GREUSSAY 76a, CHAILLOUX 78a] développés en France à l'Université de Paris 8 - Vincennes, implantés sur les 3 catégories de systèmes matériels (gros, mini et micro) et qui sont dotés des structures de contrôle les plus puissantes.

Le point commun de ces familles d'implémentations est d'avoir toutes une réalisation sur l'ordinateur de référence PDP-10 [DEC 78a], qui est la machine la plus utilisée dans le cadre des recherches en Intelligence Artificielle. Pour cette raison les performances de ces différentes réalisations sont aisément mesurables.

En plus du PDP-10, ces systèmes LISP n'ont été implantés que sur un nombre restreint d'autres matériels en général de taille très importante tels l'IBM série 360/370 [BOLCE 68, HAFNER 74], l'UNIVAC 1100 ou l'IRIS-80 pour lequel on dénombre 3 réalisations en France, le TLISP à l'Université Paul Sabatier de TOULOUSE [DURIEUX 78], le RLISP au laboratoire IMAG de GRENOBLE [LUX 78] et SIRLISP à l'E.N.S.T de Paris [COILLAND 79].

LISP 1.5 et LISP 1.6, hormis des réalisations-jouets à fins pédagogiques, sont à présent tombés en désuétude.

MACLISP et INTERLISP, trop dépendant de la machine PDP10 (et même du système d'exploitation utilisé respectivement ITS et TENEX) n'ont pu être transportés dans leur intégralité sur d'autres machines *{Note 1}*.

Seul, VLISP, (du fait probablement de sa conception en Europe et de la diversité des ordinateurs qui y sont disponibles) a pu (ou a dû) être réalisé sur un grand nombre de machines très variées telles que CAE510 [GREUSSAY 72], CAB500 [WERTZ 74], T1600 [GREUSSAY 75], PDP10 [CHAILLOUX 78c], SOLAR16 [GREUSSAY 78b], 8080 et Z80 [CHAILLOUX 79a] et PDP11 [GREUSSAY 79b].

{Note 1} En date de Septembre 1978, MACLISP ne tournait encore qu'avec beaucoup de difficultés, en ce qui concernait ses aspects-système évolués, au laboratoire d'Intelligence Artificielle de Stanford, sous système WAITS [GREUSSAY 78a].

La diversité de toutes ces réalisations a entrainé la naissance d'histoires (et d'historiens) de LISP, apportant de précieuses informations sur l'évolution naturelle des langages de programmation *{Note 1}* [McCARTHY 78] , [WHITE 78], [STOYAN 78a,78b].

Les recherches actuelles de construction des systèmes LISP s'élaborent pour l'essentiel selon les 4 axes suivants :

- implémentation sur des ordinateurs universels classiques tel le VAX-11 [DEC 78e] ou le S-1 [HAILPERN 79].
- implémentation sur des ordinateurs spécialisés, voire des micro-ordinateurs, à ressources limitées [CHAILLOUX 78a, TAFT 79].
- réalisation de machines spécialisées [GREENBLATT 74, KNIGHT 74, SHIMADA 76, LISPMACHINE 77, LECOUFFE 77, TAKI 79, WEINREB 79].
- apparition d'unités centrales, en technologie V.L.S.I. [MEAD 80], capables d'interpréter directement LISP à une translation près du langage source en sa représentation arborescente, dont le répertoire des sommets constitue un jeu d'instructions d'interprétation directe de LISP (les CHIP-LISP) [STEELE 79, HOLLOWAY 80].

{Note 1} Par opposition aux cas de langages à génération spontanée ou à comités [HORNING 79], à définition discrétionnaire.

1.2 COMMENT DECRIRE LES IMPLEMENTATIONS : la machine VCMC2.

Dès son apparition LISP a été exposé méta-circulairement *{Note 1}* afin de décrire ses propres implémentations.
Cette utilisation de LISP, très commode, ne satisfaisait pas les implémenteurs experts, qui n'avaient à leur disposition que ces descriptions méta-circulaires accompagnées, dans le meilleur des cas, du texte d'une implémentation particulière. Tous les problèmes réels d'implémentation en machine étaient soit totalement ignorés (dans le cas où LISP était utilisé à sa propre description) soit entièrement orientés vers une machine particulière.

Pour illustrer cette situation, nous utiliserons dans cette section, une succession progressive de descriptions du module classique EVLIS : nous partirons de sa description méta-circulaire en LISP, nous la qualifierons sur des machines particulières, enfin nous en donnerons la description dans notre modèle de référence.

Voici la description de la fonction EVLIS, en LISP classique, telle quelle est donnée dans les manuels de référence :

LISP classique

```
(DE EVLIS (L)
   (COND
     ((NULL L) NIL)
     (T (CONS (EVAL (CAR L)) (EVLIS (CDR L))))))
```

Une telle description purement méta-circulaire était déjà remise en question par John McCARTHY, le créateur du langage : il introduisit deux langages LISP, un langage algorithmique (le Méta-langage) qui utilisait des M-expressions et un langage de programmation (le LISP tel qu'on le parle encore) qui mettait en jeu les S-expressions [McCARTHY 62]. Ce double langage permettait de distinguer les données décrites en S-expression des programmes décrits sous forme de M-expressions [PERROT 79].

{Note 1} LISP est spécifiable par un interprète dit méta-circulaire : un tel interprète, écrit en VLISP, ramasse en une seule description la spécification du langage et celle de l'interprète lui-même (voir la description de l'évaluateur VLISP-10 en VLISP-10 donné dans [CHAILLOUX 78c] page 20).

La traduction du langage algorithmique vers le langage de programmation LISP devait s'effectuer automatiquement *{Note 1}*. on sait aujourd'hui que seul le langage de programmation a survécu.

Voici donc la description de la fonction EVLIS en utilisant la notation sous forme de M-expressions :

M-expressions

```
evlis[l]=[null[l]→NIL;T→cons[eval[car[l]];evlis[cdr[l]]]]
```

{Note 1} c'est l'absence des caractères spéciaux [,], et → sur la perforatrice de cartes IBM026 qui a empêché l'utilisation systématique des M-expressions.

Si on considère qu'une implémentation particulière constitue une description de référence on se heurte aux deux difficultés que sont :

1) l'opacité inhérente à la méconnaissance préliminaire de l'ordinateur source

2) la non-généralité chronique de ce type de description.

Voici donc cette même fonction telle qu'elle est codée dans le langage machine du PDP10 [CHAILLOUX 78c] :

Langage machine PDP-10

```
EVLIS:
    JUMPE   A1,VPOPJ        ; pas d'argument
    HRRZ    A2,MEM(A1)      ; A2 ← (CDR A1)
    PUSH    P,A2            ; qui est sauvé
    PUSHJ   P,EVALCA        ; évalue la 1ère val.
    HLRZ    A1,MEM(A1)      ; A1 ← (CONS A1)
    EXCH    A1,MEM(FREE)
    EXCH    FREE,A1
    POP     P,A2            ; récupère le reste
    CAMGE   A2,BLIST        ; au moins 2 ?
    JRST    VPOPJ           ; non : fini
    PUSH    P,A1            ; sauve le 1er doublet
    PUSH    P,A1            ; sauve le dernier
    PUSH    P,A2            ; sauve le reste
    MOVEI   A1,(A2)         ; A1 ← les arguments
EVLIS1:
    HRRZ    A2,MEM(A1)      ; A2 ← (CDR A1)
    MOVEM   A2,(P)          ; dans le sommet de la pile
    PUSHJ   P,EVALCA        ; évalue l'argument suivant
    HLRZ    A1,MEM(A1)      ; A1 ← (CONS A1)
    EXCH    A1,MEM(FREE)
    EXCH    FREE,A1
    MOVE    A2,-1(P)        ; A2 ← le dernier doublet
    HRRM    A1,MEM(A2)      ; (RPLACD A2 A1)
    MOVEM   A1,-1(P)        ; sauve le dernier doublet
    MOVE    A1,(P)          ; A1 ← le reste
    CAML    A1,BLIST        ; il reste des éléments ?
    JRST    EVLIS1          ; oui.
    SUB     P,[3,,3]        ; non : nettoie la pile .
    MOVE    A1,1(P)         ; A1 ← 1er doublet
VPOPJ:
    POPJ    P,              ; et voilà.
```

Le point 1) est illustré ici par l'instruction MOVEI, utilisée à la ligne 15, qui emploie classiquement la propriété d'adressage suivante :

immédiat + indexé = direct

et permet un gain substanciel de temps :

Temps d'exécution des instructions		
MOVE	A1,A2	1.14 micro-sec
MOVEI	A1,(A2)	0.62 micro-sec

sur l'unité centrale PDP 10 KI [DEC 78a].

On notera également la confusion conceptuelle introduite par l'utilisation du pointeur de pile P comme registre d'index, pointeur qui ira même (voir la ligne 30) jusqu'à être employé à l'extérieur de la zone pile.

La non-généralisation de cette description est malheureusement une conséquence de l'utilisation raisonnablement experte d'une machine particulière.

Nous devons aller au delà de ces 3 représentations pour aborder, décrire correctement, résoudre enfin les véritables problèmes d'implémentation en machine :

- la représentation en LISP classique montre assez bien l'intention de la fonction mais ignore complétement les problèmes d'allocation des ressources, souvent limitées, ainsi que la gestion de la récursion.

- la représentation en M-expressions tout en évoquant approximativement une notation algébrique présente les mêmes inconvénients que la représentation en LISP et introduit un niveau supplémentaire de traduction.

- la représentation en langage machine PDP10 fait surgir de nouveaux problèmes locaux tels que l'accès aux demi-mots de la machine voire l'utilisation du pointeur de pile comme registre d'index. La description en langage machine est alors supplantée par l'expérience nécessaire à la mise en jeu correcte d'adressages et de jeux d'instructions particuliers.

Nous proposons donc de représenter l'implémentation au moyen de programmes d'une machine de référence, la machine VCMC2, descendante directe de la machine VCMC1 [CHAILLOUX 78b], et indépendante des représentations internes des objets eux-mêmes. C'est de cette description que naissent naturellement les incarnations opérationnelles particulières de VLISP sur les ordinateurs les plus diversifiés. Cette machine, décrite au Chapitre 3, est simulée en VLISP-10 sur ordinateur PDP-10. Le texte de ce simulateur est donné à l'appendice B.

La description de VLISP très concise, qui en découle dégage les caractéristiques fondamentales de l'implémentation de modules tels que EVLIS en machine, i.e. :

1) les actions de base (test par rapport à NIL, CDR, CONS ...)

2) la gestion de la récursion par utilisation d'une pile et d'un accumulateur.

3) le mécanisme du sauvetage et de la restauration du reste de la liste et du résultat intermédiaire au travers de la pile.

4) l'internalisation totale des fonctions interfaces : entrées/sorties et traces (ce que ne pouvait pas faire la notation FILTRE de [GREUSSAY 76b]).

Voici donc la description, en VCMC2, de l'implémentation en machine de la fonction EVLIS :

La fonction EVLIS décrite en VCMC2

```
EVLIS:  TNIL    A1,,[RETURN]
        CDR     A1,TST,[CALL (EVCAR)]
        XTOPST  A1,,[CALL (EVLIS)]
        CONS    TST,A1,[RETURN]
```

Cette représentation générale atteint le degré de précision suffisant pour :

1) appréhender la véritable structure en actions de base de l'implémentation (ce que ne pouvait pas faire la description d'INTERLISP *{Note 1}*

2) expliciter totalement la gestion de la pile de récursion en unifiant dans l'adressage la mise en jeu des accumulateurs et du cache de pile.

3) indiquer la circulation dans la pile des états d'évaluation intermédiaires du contenu initial du registre A1

4) préciser enfin la structure de contrôle par attachement à chaque instruction d'une continuation complexe.

{Note 1} la notation utilisée dans la description de [MOORE 76] ressemble à la notation sous forme de M-expressions et ne rend pas compte de la gestion de la récursion et des résultats intermédiaires. Sachant que EVLIS est la version SUBR du module LIST (de type FSUBR) [MOORE 76 pp. 11] ne peut décrire cette dernière qu'elliptiquement par le dispositif typographique des points de suspension :

```
LIST[x1;x2;...xk]
   Return CONS[x1;CONS[x2;...CONS[xk;NIL]...]]
```

1.3 EFFICACITE ET PUISSANCE DE L'INTERPRETATION.

Du fait de sa représentation des objets et des fonctions, notre modèle permet de réduire considérablement le temps d'interprétation et de disposer d'un grand nombre de structures nouvelles, qui facilitent l'écriture des programmes, en améliorent la lisibilité et en diminuent sensiblement la taille.

Voici la définition de la fonction de FIBONACCI utilisée pour des comparaisons de vitesse sur plusieurs interprètes LISP fonctionnant sur PDP10 :

```
(DE FIB (n)
    (COND
       ((ZEROP n) 1)
       ((EQ n 1) 1)
       (T (PLUS (FIB (SUB1 n)) (FIB (DIFFER n 2))))))

(FIB 20)  ☞  10946
```

Voici les temps relevés sur MACLISP, INTERLISP, VLISP-10 et VLISP-11 *{Note 1}* pour le calcul de l'appel **(FIB 20)** :

[Temps en secondes]			
MACLISP	INTERLISP	VLISP-10	VLISP-11
48.66	105.78	13.34	35.2

{Note 1} MACLISP utilise dans le cadre de ce test une unité centrale PDP10 KA-10 et des mémoires à 1.8 micro-secondes [WHITE 76], INTERLISP une unité centrale PDP10 KA-10 et des mémoires à 1.0 micro-secondes, VLISP-10 une unité centrale PDP10 KI-10 et des mémoires à 1.0 micro-secondes. VLISP-11 n'a à sa disposition qu'une micro-unité centrale LSI 11/02.

La rapidité remarquable de l'interprétation de notre modèle est dûe :

1) à l'utilisation intensive des propriétés naturelles des atomes et principalement du type associé à chaque fonction. Le type d'une fonction LISP est déterminé par le langage dans lequel elle est écrite, le mode de passage de ses arguments (les arguments sont ou ne sont pas évalués) ainsi que le nombre de ses arguments. Cette typologie est abondamment décrite au chapitre 4. Ce type est directement accessible, ce qui permet de réaliser en même temps et le test du type et l'appel de la routine associée, au moyen d'un seul branchement indirect indexé. En terme de syntaxe, VLISP utilise donc l'appel par nom des fonctions.

2) au fait que l'interprète ne réalise plus aucun CONS pour ses besoins propres. L'utilisation exclusive d'une pile commune de données et de contrôle évite de créer des structures intermédiaires (sous forme de cellules de liste) durant le processus d'évaluation. Le nombre de récupérations de la mémoire dynamique se trouve ainsi réduit aux seuls besoins de l'utilisateur *{Note 1}*.

3) à l'interprétation itérative des fonctions récursives de type NPR et CPR [GREUSSAY 76c].

4) à l'utilisation des structures de contrôle de VLISP qui permettent une écriture plus concise et une interprétation plus rapide. En particulier :

- l'opérateur point-fixe du λ-calcul [ROBINET 78], représenté par la fonction **SELF** de [GREUSSAY 77]

  FIBONACCI se reécrit alors commodément :

```
(DE FIB (n)
    (IF (ZEROP n) 1
       (IF (EQ n 1) 1
          (PLUS (SELF (SUB1 n))
                (SELF (DIFFER n 2))))))
```

- les fonctions d'échappement **ESCAPE**. Introduites par [GREUSSAY 76b], elles permettent d'associer dynamiquement des fonctions d'échappement à des atomes, réalisant ainsi des sorties non-locales.

{Note 1} La gestion de la mémoire en LISP est automatique et dynamique ce qui assure une utilisation optimum de la place mémoire mais oblige à construire des modules d'accès et d'allocation (le module CONS) spécialisés ainsi qu'un récupérateur de mémoire, dispositif qui ralentit l'interprète. Ces problèmes de représentation des objets sont abordés au chapitre 2.

5) enfin à de nouvelles structures de contrôle. Ces nouvelles structures de contrôle vont pouvoir réaliser :

- la description de fonctions anonymes (i.e. qui ne sont pas associées à des noms) en utilisant la forme **INTERNAL**.

- les sorties locales à une fonction en utilisant la fonction **EXIT**.

- la redéfinition dynamique de fonctions de n'importe quel type en utilisant la primitive **WHERE**.

- la définition d'une nouvelle classe de fonctions, les variables fonctions, apportant une solution nouvelle au problème des variables internes.

1.3.1 Description de fonctions anonymes

Notre modèle permet de décrire des fonctions anonymes (non associées à des symboles atomiques) de n'importe quel type au moyen d'une des deux formes **INTERNAL** suivantes :

```
(INTERNAL type adresse-mémoire)
(INTERNAL type fonction)
```

La première forme permet de décrire des fonctions écrites en langage machine, et la seconde des fonctions écrites en VLISP.

Ces deux nouvelles formes généralisent l'ancienne utilisation de la forme LAMBDA (qui a été gardée par souci de compatibilité et de simplification de l'écriture des EXPR anonymes) pour tous les types de fonctions et permettent ainsi de limiter l'utilisation des noms associés aux fonctions.

Cette limitation des noms réduit d'autant plus l'espace mémoire nécessaire au stockage des symboles atomiques et accroit la lisibilité du programme qui ne contient plus que les noms indispensables.

1.3.2 Sorties locales

La forme **EXIT** permet de sortir de la dernière fonction invoquée de type EXPR, FEXPR ou MACRO. La valeur retournée est évaluée dans l'environnement précédant exactement le retour de cette fonction donc évaluée en position terminale. Cette propriété est utilisée pour forcer un traitement de récursion terminale *{Note 1}* ce qui n'était pas le cas des fonctions **RETURN** des anciennes formes **PROG** *{Note 2}*.

Voici la méta-description de la fonction **MEMBER** qui utilise la structure de contrôle itérative **WHILE**.

```
(DE MEMBER (a l)
   (WHILE L
      (IF (EQUAL (CAR l) a)
         (EXIT l)
         (NEXTL l))))
```

L'appel de la fonction **EXIT** permet une sortie extraordinaire du **WHILE**.

{Note 1} Ces appels récursifs terminaux sont interprétés d'une manière itérative beaucoup plus efficiente.

{Note 2} La forme **PROG** *a été abandonnée dans notre modèle (ainsi que ses fonctions associées* **GO**, **GOTO** *et* RETURN*) car les nouvelles fonctions de sorties locales et non-locales sont à la fois plus rapides (en effet le balayage préliminaire de tous les corps de* **PROG** *pour construire la table des étiquettes a disparu) et plus générales (avec la possibilité de retourner directement d'un nombre quelconque d'appels).*

1.3.3 Définitions dynamiques de fonctions

La forme **WHERE** permet de définir dynamiquement de nouvelles fonctions et de rendre fluides *{Note 1}* les fonctions elles-même.

Voici la syntaxe de cette nouvelle forme :

```
(WHERE (<nom> <fval>)  <s1> ... <sN>)
```

Le premier argument du **WHERE** est une définition temporaire d'une fonction **<fval>** associée au nom **<nom>**. Le reste des arguments du **WHERE** i.e. **<s1>** ... **<sN>** est un corps dans lequel la définition précédente est active. Au sortir du **WHERE**, la définition associée au nom **<nom>** disparait et sa définition antérieure (s'il en possédait une) est restaurée.

Voici la fonction de traçage d'une courbe Dragon [GARDNER 67, SCHOETTL 75] en LOGO-LISP [WERTZ 79] dans laquelle la fonction **tourne** est redéfinie dynamiquement.
Les fonctions **<avance>**, **<droite>** et **<gauche>** sont des primitives LOGO :

```
(DE tourne () (<droite> 90))

(DE dragon (n l)
   (IF (ZEROP n)
      (<avance> l)
      (WHERE (tourne '(() (<gauche> 90)))
          (dragon (SUB1 n) l))
      (tourne)
      (WHERE (tourne '(() (<droite> 90)))
          (dragon (SUB1 n) l))))
```

{Note 1} Une variable fluide est une variable libre liée dynamiquement.

1.3.4 Les variables fonctions

Notre modèle propose une solution au problème de l'accès aux variables internes *{Note 1}* de l'interprète. En effet, un choix crucial d'implémentation se pose.
Il n'est que trop tentant d'utiliser des variables VLISP pour accéder à ces variables internes. Appelons OBASE la variable interne qui contient à tout moment la base de numération des nombres en sortie. Une affectation incorrecte de cette variable (par exemple avec une valeur négative), détruirait toute possibilité future d'impression de nombres.
Une deuxième stratégie consiste à utiliser une fonction associée à chaque variable interne. Cette fonction va contrôler la validité des valeurs affectées à la variable. C'est ce qu'on entend par variabilisation ou fonctionnalisation de l'accès.

C'est cette dernière stratégie qui a été adoptée dans notre modèle. De plus celui-ci propose une nouvelle classe de fonctions, les variables fonctions, qui permettent dans une stratégie de fonctionnalisation des variables internes de lier dynamiquement les valeurs de ces variables internes, retrouvant ainsi toutes les propriétés des variables.
En lecture ces variables fonctions n'ont pas d'argument et en écriture elles possèdent un argument (évalué) qui est la nouvelle valeur de la variable fonction.

```
(variable-fonction)              :   lecture
(variable-fonction  valeur)      :   écriture
```

La nouvelle structure LETF réalise la liaison dynamique des variables fonctions. Elle possède une syntaxe identique à la macro LET :

```
(LETF (variable-fonction  valeur) corps-du-LETF)
```

{Note 1} rappelons que ces variables internes sont les mots mémoires de travail de l'interprète et qu'il est souhaitable que l'utilisateur ait accès à certaines de ces variables pour faciliter l'utilisation du système et en augmenter la puissance.

Cette forme est exécutée en 4 temps :

1) appel de la variable fonction en lecture (sans argument). La valeur retournée est conservée.

2) appel en écriture de la variable fonction avec l'argument transmis au **LETF**.

3) exécution en séquence du corps du **LETF** qui calcule la valeur retournée par la forme **LETF**.

4) re-appel en écriture de la variable fonction avec en argument son ancienne valeur conservée en 1).

Ce type de liaison (de même que celle des variables) est réalisée automatiquement et en particulier le point 4) est effectuée en cas de sortie extraordinaire par un **EXIT** ou un **ESCAPE** enveloppant.

L'utilisation de ces variables fonctions permet donc d'associer à des variables des fonctions qui réalisent des contrôles d'accès dynamiques tant en lecture qu'en écriture.

Voici un exemple d'utilisation de la variable fonction **OBASE** qui contrôle la base de conversion des nombres en sortie. La fonction **PRINT-base-x** imprime son 2ème argument **n** dans la base de numération **x** fournie en 1er argument :

```
(OBASE)  ☞  10

(DE PRINT-base-x (x n)
    (LETF (OBASE x)
       (PRINT n)))

(PRINT-base-x 16 1000)  ☞  3E8

(OBASE)  ☞  10
```

1.4 LA SOUPLESSE D'UTILISATION.

Un certain type de programmation-système consiste à implanter, valider (ou invalider), mesurer, modéliser de nouveaux types d'architectures logicielles ou matérielles (par simulateur), à la limite de l'élaboration de modèles en Intelligence Artificielle.
Cette programmation se fera tout naturellement dans les conditions les plus interactives et les plus expérimentales possibles, et nécessitera des outils permettant la réalisation la plus rapide du modèle à valider. Cette activité demande donc un langage puissant, très facilement et très rapidement mis en oeuvre, car les temps de vie des différents modèles sont extrêmement brefs et soumis à des modifications incessantes.

Ainsi notre système ne mésestime pas l'expertise de ses utilisateurs. Il s'agit d'un système utilisable et utilisé, illustrant la nécessité, parfois sous-estimée, de ne pas entraver l'expertise raisonnable de ses utilisateurs (par exemple en ne l'encombrant pas de contrôles statiques contraignant à une programmation médiocre) mais, tout au contraire, de leur fournir le maximum de pouvoir et de souplesse expressive.

L'utilisation réelle d'une tel système nécessite la faculté de contrôler par niveau le fonctionnement de l'interprète, ce qui amène tout naturellement à l'idée d'évaluateurs multiples selon les circonstances du calcul. Les outils de contrôle de chronologie, traces et erreurs qui vont être décrits en sont une illustration.

1.4.1 La notion de CHRONOLOGIE

Notre modèle dispose d'une multitude d'évaluateurs potentiels différenciés par un numéro d'ordre de création, un numéro de chronologie. L'accès à cette chronologie est réalisée au moyen de la variable fonction CHRONOLOGY.

Ces évaluateurs peuvent être appelés :

- par l'utilisateur au moyen de la fonction interprète EVAL dont le deuxième argument est le numéro de CHRONOLOGIE que l'on veut voir affecté à cette évaluation :

```
(EVAL expression chronologie)
```

- soit automatiquement par l'interprète, il s'agit alors d'interruptions.
  Ces interruptions sont déclenchées par des événements externes (horloges ou périphériques) ou par l'interprète lui-même dans les cas suivants :
  - appel de la boucle principale du système (TOPLEVEL)
  - erreur à l'interprétation (ERROR)
  - trace de l'évaluateur (STEPEVAL)
  - ligne pleine en sortie (EOL)
  - apparition d'une fin de fichier d'entrée (EOF)

Le traitement d'une interruption se fait par invocation d'une fonction propre à chaque type d'interruption, dans un nouvel évaluateur dont la chronologie est la chronologie précédente incrémentée d'une unité, i.e. :

```
(EVAL '(fonction associée à l'IT ....) (ADD1 (CHRONOLOGY)))
```

La valeur retournée par la fonction associée à l'interruption devient la valeur de l'interruption.

Il existe de plus une fonction de sortie extra-chronologie, la fonction **EXITCHRONOLOGY** qui réalise la sortie de la chronologie courante et retourne une valeur au créateur de cette chronologie. Cette fonction possède la même syntaxe que la fonction de sortie locale à une fonction, la fonction **EXIT**.

1.4.2 Les erreurs

Notre modèle ne provoque pas l'abandon du travail en cas de détection d'erreurs. Lorsqu'il se produit des états d'indétermination dans un évaluateur, celui-ci va dynamiquement créer un nouvel évaluateur. Sa tâche est de tenter de lever l'indétermination ou tout au moins de retourner une valeur à l'évaluateur interrompu.

Lorsque ces états d'indétermination sont détectés par un interprète (par exemple lors d'une consultation d'une variable non définie), une fonction spéciale, la fonction **ERROR**, est automatiquement invoquée avec comme arguments les objets indeterminés, dans un évaluateur différent (dont la chronologie est égale à la chronologie antérieure incrémentée d'une unité). La valeur ramenée par cette invocation est utilisée pour lever l'indétermination de l'interprète interrompu.

Cette fonction **ERROR** (comme toutes les fonctions d'interruptions) peut être redéfinie en VLISP afin d'utiliser des sytèmes complexes de corrections automatiques tel que le système PHENARETE de [WERTZ 78], des outils sophistiqués d'aides-à-la mise au point tel que le méta-évaluateur CAN de [GOOSSENS 77, 79], des analyseurs [WERTZ 77] des traces ou des steppers [GREUSSAY 79a]

Voici un exemple de la redéfinition de la fonction **ERROR** qui permet de construire une fonction testant si une forme peut être évaluée sans erreur par EVAL. Dans le cas ou EVAL produirait une erreur cette fonction retourne la valeur NIL, et dans le cas contraire elle retourne la valeur de l'évaluation. Cette dernière valeur est incluse en premier élément d'une liste pour pouvoir distinguer la valeur NIL correspondant à une évaluation de cette même valeur indiquant une erreur à l'évaluation.

```
(DE EVAL-teste-si-erreur (forme)
    (WHERE (ERROR '(() (erreur)))
      (ESCAPE erreur
          [(EVAL forme)])))
```

1.4.3 Les traces

Une des facilités majeures de notre modèle est d'avoir un mécanisme interne d'observation sélective de l'évaluateur lui-même en fonctionnement. Ce mécanisme permet d'appeler la fonction **STEPEVAL** (qui peut être bien entendu redéfinie en VLISP) avec comme argument la forme qui doit être évaluée, à chaque appel interne de l'évaluateur.

Ce mode trace est activé en donnant une valeur non-NIL au 3ème argument de la fonction interprète EVAL :

```
(EVAL expression chronologie trace)
```

Voici comment réaliser une trace de tous les appels de EVAL.

```
(DE TRACEVAL (exp)
    (WHERE
        (STEPEVAL (forme)
            (PRINT '→ forme)
            (SETQ IT (EVAL forme (SUB1 (CHRONOLOGY)) T))
            (PRINT '← IT))
        (EVAL '(STEPEVAL exp) (ADD1 (CHRONOLOGY)))))

; Fonction de test : dernier élément de l ;

(DE FOO (l)
    (IF (NULL (CDR l)) (CAR l) (SELF (CDR l))))

; Appel de la trace ;

(TRACEVAL '(FOO '(A B)))

→   (FOO '(A B))
→   '(A B)
←   (A B)
→   (IF (NULL (CDR l)) (CAR l) (SELF (CDR l)))
→   (NULL (CDR l))
→   (CDR l)
→   l
←   (A B)
←   (B)
←   NIL
→   (SELF (CDR l))
→   (CDR l)
→   l
←   (A B)
←   (B)
```

```
→  (IF (NULL (CDR l)) (CAR l) (SELF (CDR l)))
→  (NULL (CDR l))
→  (CDR l)
→  l
←  (B)
←  NIL
←  T
→  (CAR l)
→  l
←  (B)
←  B
←  B
←  B
←  B
←  B

☞  B
```

Une des difficultés classiques de la réalisation du mécanisme de trace est posée par l'interférence entre la fonction traçante et la fonction tracée en particulier en cas d'utilisation des constructions **SELF** et **EXIT** : en effet dans la 11ème ligne de trace l'appel **(SELF (CDR l))** doit utiliser la fonction tracée (i.e. la fonction **FOO**) et non pas la dernière fonction appelée (i.e. la fonction **STEPEVAL** elle-même). De même l'évaluation d'une forme **EXIT** peut se rapporter soit au programme traçant soit au programme tracé. Pour lever toute ambiguité à l'évaluation des formes **SELF** et **EXIT**, notre modèle utilise la dernière fonction appelée dans la chronologie courante. La fonction traçante et la fonction tracée sont évaluées dans des chronologies différentes, ce mécanisme permettant également de réaliser des traces méta-circulaires *{Note 1}*.

Bien entendu des traces plus élaborées (comprenant des renfoncements correspondants aux niveaux d'imbrications des appels ou des numérotations de lignes par niveau et des possibilitées d'exécution incrémentales) peuvent être construites. Ces traces sont développées au chapitre 7.

Le fait de pouvoir récupérer TOUS les appels internes de l'évaluateur permet, outre les traces, une modification complète de celui-ci.

{Note 1} KNUTH [KNUTH 69] propose page 211 l'exercice suivant : "6. [40] Design a trace routine which is capable of tracing itself, in the sense of exercice 4; i.e., it should print out the steps of its own program at slower speed, and that program will be tracing itself at still slower speed, ad infinitum until memory capacity is exceeded." Notre mécanisme de trace permet de ne tracer qu'un seul niveau de la fonction de trace.

Voici en exemple la fonction **EVAL-NIL-si-UNDEF** qui se comporte comme la fonction interprète standard EVAL mais qui ne provoque pas d'erreur à l'évaluation d'une variable indéfinie, se contentant uniquement de donner la valeur NIL à la variable et d'imprimer un message d'avertissement :

```
(DE EVAL-NIL-si-UNDEF (expression)
   (WHERE
      (STEPEVAL '((forme)
         (IF (OR (LISTP forme) (NUMBP forme) (BOUNDP forme))
             (EVAL forme () T)
             (PRINT "Je donne la valeur NIL à :" forme)
             (SET forme NIL))))
      (STEPEVAL expression)))

Voici quelques utilisations de cette fonction en supposant
que les variables VARF001, VARF002 et VARF003 sont indéfinies

?    (EVAL-NIL-si-UNDEF 'VARF001)
        Je donne la valeur NIL à : VARF001
☞   NIL

?    (EVAL-NIL-si-UNDEF '['A VARF002 'B VARF003])
        Je donne la valeur NIL à : VARF002
        Je donne la valeur NIL à : VARF003
☞   (A NIL B NIL)
```

On notera dans cette fonction la redéfinition dynamique de la fonction **STEPEVAL**, qui permet de n'effectuer le test de variable indéfinie que dans la portée dynamique de la fonction **EVAL-NIL-si-UNDEF**.

1.5 LES SYSTEMES VLISP REALISES.

Une des caractéristiques les plus importantes de notre modèle est d'être très rapidement et très facilement implémenté sur les matériels actuels. Trois réalisations récentes montrent la diversité des machines pouvant recevoir ce système :

1) le système VLISP-10 [CHAILLOUX 78c], est réalisé sur l'ordinateur PDP KI 10 de Digital Equipment Corporation [DEC 78a], sous moniteur TOPS10 6.03 [DEC 78b], SAIL [HARVEY 74, FROST 75] et IRCAM [FROST 77]. C'est le plus rapide et le plus puissant système VLISP existant. Il est disponible dans les centres suivants : l'I.R.C.A.M. et le C.I.T.I.2 en France, et EDIMBOURG, TURIN et STANFORD SAIL à l'étranger.

2) le système VLISP-11 [GREUSSAY 79b], est réalisé sur l'ordinateur PDP 11 également de Digital Equipment Corporation [DEC 75a] sous système RT11-V03 [DEC 78c]. Ce système fonctionne actuellement sur les processeurs PDP11 et sur les micro-processeurs LSI11 [DEC 78d] toujours sous système RT11-V03.

3) le système VLISP-8 [CHAILLOUX 79a], est conçu pour le micro-processeur Intel-8080 [INTEL 77a] et Zilog-80 [ZILOG 78]. Il est disponible sous le système de développement ISIS-II [INTEL 77b] et TRS80 Level II [TRS 78].

Chacun de ces systèmes représente une catégorie matérielle spécifique : gros système pour le PDP-10, mini-ordinateur pour le PDP-11 et micro-ordinateur pour le 8080. Les différences entre ces matériels sont considérables. Nous évoquerons les points suivants :

- la taille des mémoires. Le PDP10 permet d'accéder à 256k de 36 bits, le PDP11 à 28k de 16 bits *{Note 1}* et le 8080 à 64k de 8 bits.

- la vitesse d'exécution des instructions. Le transfert dans un registre du contenu d'un mot mémoire demande (indépendemment du nombre de bits transférés) 1.06 micro-sec dans un PDP10-KI, 3.2 micro-sec dans un PDP11/40 et 4.5 micro-sec dans un 8080-2Mhz.

{Note 1}* *ce nombre inhabituel de k mémoire (28k) provient du fait que les entrée/sorties sont réalisées en memory mapped I/O sur 4k mots, qui ajoutés aux 28k mémoire conduit bien à un espace adresse de 32k mots.

- la puissance des instructions. Les jeux d'instructions de ces machines n'ont pas la même puissance ce qui amène à utiliser un nombre d'intructions plus ou moins grand. Par exemple pour la réalisation de l'allocation et de la construction d'un doublet de liste (le module CONS), il suffit de 2 instructions du PDP10 (qui occupent chacune un mot mémoire de 36 bits) alors qu'il faut un sous-programme de 9 instructions pour le PDP11 (chaque instruction nécessite de 1 à 3 mots de 16 bits) occupant un total de 12 mots pour l'appel et l'exécution du sous-programme. Quant aux performances du 8080 elles sont encore plus mauvaises puisqu'il faut un sous-programme de 21 instructions, qui nécessitent chacune de 1 à 3 mots de 8 bits) occupant un total de 30 mots mémoire.

Malgrés leurs différences, il a été possible d'implanter sur toutes ces machines un système VLISP, tel celui que nous décrire dans cette étude. Ces différents systèmes ne se différencient que par des vitesses d'exécution et des espaces mémoire différents.

Enfin, le champ d'application du système VLISP est extrêmement vaste. Ses principales applications récentes recouvrent les domaines suivants :

- l'amélioration et la correction de programmes avec le système PHENARETE [WERTZ 79].

- la méta-interprétation de programmes récursifs avec le système CAN [GOOSSENS 79].

- les algorithmes d'unification [HULLOT 79].

- la synthèse de programmes à partir d'exemples avec le système SISP [JOUANNAUD 77].

- la compréhension de programmes avec le système RAINBOW [GREUSSAY 79a].

- la synthèse d'images colorées telle qu'elle est pratiquée par le groupe Art et Informatique de l'Université de Vincennes [AUDOIRE 76], [HUITRIC 76]

- l'aide à l'éducation des enfants retardés avec le système LOGO/LISP [WERTZ 79].

- les outils de conception de machines [CHAILLOUX 78b, CHAILLOUX 79b].

1.6 PLAN DE L'ETUDE.

Notre étude se découpe de la façon suivante :

Le chapitre **2** traite de la représentation des objets de base manipulés par les interprètes VLISP, en particulier de celle des listes, des nombres et des symboles atomiques.

Le chapitre **3** contient la description de la machine référentielle VCMC2, utilisée pour construire notre modèle d'implémentation. Nous y donnerons de nombreuses statistiques d'utilisation de cette machine. L'appendice **A** contient une description résumée de cette machine et l'appendice **B** le texte du simulateur de cette machine écrit en VLISP 10.

Le chapitre **4** décrit les mécanismes fondamentaux de l'interprétation de VLISP, en particulier l'accès aux variables et la liaison des arguments. Cette description utilise très largement la machine référentielle VCMC2 ce qui permet d'évaluer les différentes possibilités.

Le chapitre **5** traite de la conversion des représentations externes des objets VLISP en représentation interne. L'appendice **C** contient le texte complet des fonctions d'entrée du modèle de référence écrites en VCMC2.

Le chapitre **6** traite des problèmes de représentation externe des objets VLISP et en particulier ceux relatifs à l'édition des structures VLISP normales, circulaires ou partagées. L'appendice **D** contient le texte des fonctions écrites en VLISP réalisant les éditions des programmes VLISP. L'appendice **E** contient le texte des fonctions de sortie du modèle de référence écrites en VCMC2.

Enfin le chapitre **7** est consacré aux concepts avancés de l'interprétation du langage. En particulier y seront abordés les problèmes suivants :
- l'utilisation de la pile comme structure de contrôle hétérogène
- l'interprétation itératives de certaines fonctions récursives
- la création de fonctions de contrôle local comme **SELF** et **EXIT**
- la création de fonctions dynamiques (la forme **WHERE**)
- la création de fonctions dynamiques de contrôle non-local (la forme **ESCAPE**)
- la création de variables fonctions (la forme **LETF**)

Ce dernier chapitre traite également des différents états du système et apporte une solution au problème du contrôle des erreurs, de la trace et de l'exécution incrémentale grâce à la notion de chronologie d'activation du système.

L'appendice **F** contient la liste complète de l'interprète tel qu'il a été décrit aux chapitre **4**, et **7**.

Enfin l'appendice **G** contient la liste des tests qui ont servi à mesurer les performances de notre système.

CHAPITRE 2

LA REPRESENTATION DES OBJETS VLISP

Les trois réalisations d'interprètes VLISP décrites dans cette étude ainsi que le modèle d'implémentation qui en découle manipulent des objets de type *{Note 1}*
- symboles atomiques,
- nombres,
- listes,

qui sont les trois grandes classes d'objets rencontrés dans les systèmes LISP.

Ce chapitre décrit la représentation physique en machine de ces objets. L'utilisation de la machine VCMC2 permettra par la suite de se dégager des représentations physiques.

Les objets LISP sont riches et originaux du fait de leur structure et de leur gestion :

- ils sont trop complexes pour être rangés dans un seul mot machine, et sont en général réprésentés par un pointeur (i.e. une adresse) sur la zone mémoire contenant la (ou les) valeurs de l'objet LISP. Tout accès à un objet LISP se fait donc par indirection au travers d'un pointeur.

- la gestion de la mémoire qui contient ces objets est dynamique et automatique ce qui permet de gérer l'espace mémoire de manière optimale.

{Note 1} La notion de type sera employée dans ce chapitre de manière exclusivement différentielle.

2.1 COMMENT REPRESENTER LES LISTES ET LEURS MODIFICATEURS.

2.1.1 La description des listes

Très souvent, un programmeur manipulera les listes comme des séquences d'objets. Reste qu'en LISP, les listes sont en fait des arbres binaires. Chaque noeud de l'arbre est appelé doublet (ou cellule). Les deux composants de chaque doublet peuvent faire référence à d'autres doublets ou bien à des objets terminaux d'une grande variété, les atomes. Les deux composants de chaque doublet sont appelés CAR et CDR (dites *coudère* en Bretagne). Ces deux noms charmants proviennent des mnémoniques des instructions d'une très ancienne machine l'IBM 704 sur laquelle a été implanté le tout premier LISP, le LISP 1 [McCARTHY 60a]. Les mots de cette machine comprenaient 36 bits, séparés en quatre champs dont deux de 15 bits pouvant contenir des adresses. Un doublet était représenté par un mot. Les deux instructions CAR et CDR, dont les noms signifient Content of Address part of Register et Content of Decrement part of Register, permettaient d'avoir accès aux deux composants de type adresse d'un mot [McCARTHY 78].
Par la suite, ces deux noms sont restés, et toutes les tentatives de changement de nom de ces fonctions (en MEM et REM dans le langage TREET [HAINES 69] ou en HD et TL dans le langage POP2 [BURSTALL 71]) sont jusqu'à maintenant restées vaines.

La représentation classique d'un doublet de liste consiste à utiliser deux pointeurs : le pointeur CAR et le pointeur CDR.

Nous décrirons un doublet de liste en mémoire de la manière suivante :

$$z{:}C\Big|^{x}_{y} \qquad \text{ou bien} \qquad z{:} \quad \text{par défaut}$$

Dans lequel *z:* représente le pointeur sur le doublet (son adresse), *x* le pointeur CAR et *y* le pointeur CDR.

Nous représenterons une liste par un ensemble de doublets entre accolades, et les atomes par des lettres capitales.

Ainsi nous représenterons la liste :

(A B (C D) E)

par l'ensemble de doublets :

$$\{\ s{:}C\Big|^{A}_{t},\ t{:}C\Big|^{B}_{u},\ u{:}C\Big|^{w}_{v},\ v{:}C\Big|^{E}_{NIL}\ w{:}C\Big|^{C}_{x},\ x{:}C\Big|^{D}_{NIL}\ \}$$

Cette notation nous permettra de décrire précisement l'action du module d'allocation de doublet (le module CONS) en terme de modification de doublets.

LISP comporte traditionnellement 5 primitives de manipulation de doublets de listes, 2 primitives d'accès CAR et CDR, 2 primitives de modification RPLACA et RPLACD et 1 primitive de construction CONS.

1) la primitive d'accès CAR permet d'accéder au pointeur CAR d'un doublet

(CAR v:C $\left|\begin{matrix}a\\d\end{matrix}\right.$ *)* ☞ *a*

2) la primitive d'accès CDR permet d'accéder au pointeur CDR d'un doublet

(CDR v:C $\left|\begin{matrix}a\\d\end{matrix}\right.$ *)* ☞ *d*

3) la primitive RPLACA permet de remplacer le pointeur CAR d'un doublet.

(RPLACA v:C $\left|\begin{matrix}a\\d\end{matrix}\right.$ *x)* ☞ *v:C* $\left|\begin{matrix}x\\d\end{matrix}\right.$

(RPLACA v: (CAR v:)) ☞ *v:*

4) la primitive RPLACD permet de remplacer le pointeur CDR d'un doublet

> ***(RPLACD v:C|a/d y)*** ☞ ***v:C|a/y***
>
> ***(RPLACD v: (CDR v:))*** ☞ ***v:***

5) la primitive CONS permet de construire un doublet de listes avec ses 2 composants CAR et CDR.

> ***(CONS a d)*** ☞ ***w:C|a/d***
>
> ***(CONS (CAR x:C|a/d) (CDR x:C|a/d))*** ☞ ***y:C|a/d***

La deuxième construction réalise une recopie des composants *a* et *d* du doublet ***x:*** dans un doublet neuf ***y:***.

2.1.2 La représentation physique des listes

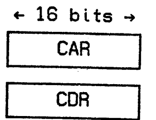
La représentation physique d'un doublet de liste est réalisée au moyen d'un ou plusieurs mots mémoire consécutifs, contenant le pointeur CAR et le pointeur CDR.

Le nombre de mots dépend de la machine utilisée et de la taille de ses mots.
Un doublet occupe :

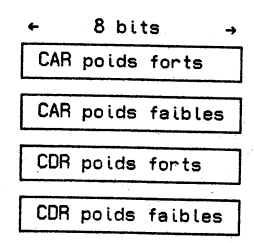
- 1 mot de 36 bits sur PDP 10 qui a des adresses sur 18 bits :

CAR	CDR
← 18 bits →	← 18 bits →

- 2 mots de 16 bits sur T1600, SOLAR ou PDP 11 dont les adresses sont sur 16 bits :



- 4 mots de 8 bits sur 8080 ou Z80 dont les adresses sont sur 16 bits :



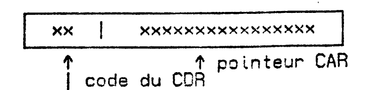
Nous connaissons à présent, grâce aux récentes études de CLARK sur les structures de listes tant statiques que dynamiques [CLARK 77, 79], d'une part la fréquence d'utilisation des différents types d'objets LISP :
- les CAR des doublets contiennent pour 70% d'éléments terminaux (symbols atomiques, nombres ou NIL) et pour 30% des listes
- les CDR des doublets contiennent pour 75% des pointeurs vers d'autres doublets, pour 20% l'atome NIL et pour 5% d'autres éléments terminaux.

et d'autre part le principe de localité des pointeurs LISP :
- près de 2 pointeurs CDR (vers des listes) sur 3 contiennent l'adresse du mot suivant.

Ceci a amené une nouvelle représentaion compacte des listes, le CDR-coding, utilisé dans 2 réalisations importantes :
- la machine LISP du M.I.T. [GREENBLATT 74, LISPMACHINE 77]
- le LISP micro-progammé sur PDP11 de [GRIGNETI 76]

et discuté dans [BAKER 78, LECOUFFE 79].

Dans cette représentation un doublet de liste occupe 1 ou 2 mots mémoire. Le 1er mot contient le pointeur CAR et un code décrivant le CDR.



Voici les différents codes possibles :

CDR-Coding

00	le CDR se trouve dans le mot suivant
01	le CDR est le mot suivant
10	le CDR est NIL
11	pointeur sur un véritable doublet

Le code **11** sert à réaliser l'indirection d'adresse en cas de déplacement d'un doublet compacté dû à une modification de la partie CDR. Cette indirection devra être prise en compte par les modules d'accès CAR, CDR, RPLACA et RPLACD.

Le gain apporté par une telle organisation est de l'ordre de 30% de l'espace. Chiffre non négligeable s'il s'agit du nombre de doublets disponibles mais chiffre encore plus avantageux s'il s'agit de la taille des adresses.

2.1.3 Gestion des listes

L'allocation des doublets de liste s'effectue par allocation de doublets dans une liste de doublets pré-alloués, la liste libre. Quand celle-ci vient à épuisement, un dispositif connu sous le nom de ***récupérateur*** (ou Garbage-collection) est invoqué avec pour mission de reconstruire une nouvelle liste libre.

En supposant que le doublet alloué fait partie d'une liste de doublets libres (que l'on appelle ***free:***), la primitive CONS vue précédemment peut être décrite au moyen des primitives CDR, RPLACA et RPLACD :

si free: contient la liste

$$free: = \{ \; x{:}C\left|\begin{matrix} e \\ y \end{matrix}\right. , \; y{:}C\left|\begin{matrix} f \\ g \end{matrix}\right. , \; \ldots , \; z: \; \}$$

$$(CONS \;\; a \;\; d\,) \quad fabrique \quad x{:}C\left|\begin{matrix} a \\ d \end{matrix}\right.$$

et la liste libre devient

$$free: = \{ \; y{:}C\left|\begin{matrix} f \\ g \end{matrix}\right. , \; \ldots , \; z: \; \}$$

Ces problèmes d'allocation et de récupération des doublets de liste ont fait l'objet de nombreuses études récentes [KUROKAWA 77], qui ont apportées des solutions au problème du récupérateur fonctionnant en parallèle. On peut citer [KNUTH 69 pp. 422] (qui attribue à MINSKY l'idée première d'un récupérateur parallèle), [STEELE 75], [DEUTSCH 76], [WADLER 76], [BAKER 78] et [DIJKSTRA 78].

2.2 LES SYMBOLES ATOMIQUES.

Les symboles atomiques sont (avec les nombres) les éléments terminaux des listes. Ils sont parfois appelés atomes littéraux pour les différencier des nombres qui, eux, sont appelés atomes numériques.

Les symboles sont définis comme un ensemble de propriétés. Une propriété est un couple composé du nom de la propriété et de sa valeur associée. Il existe 2 classes de propriétés :

- les propriétées naturelles ou innées que possèdent tous les symboles dès leur création. Il s'agit d'attributs tels leur nom externe, ou leur valeur initiale. Ces propriétés sont en nombre fixe.

- les propriétes définies par l'utilisateur qui peuvent être en nombre illimité (tout au moins limité par la taille de la machine).

Historiquement un symbole était représenté par un doublet de liste. Le CAR de ce doublet contenait un indicateur spécial *<*ATOM*>* indiquant le type terminal de ce doublet, et le CDR contenait la liste des propriétés de l'atome (ou ***P-LIST***).

symbole atomique = ***x:C***	***<*ATOM*>***	
	P-LIST	

Cette représentation permettait d'utiliser les primitives d'accès, de modification et de construction des doublets de liste, mais rendait la recherche des propriétés naturelles du symbole longues et dépendantes du nombre total de propriétés du symbole (i.e. de la taille de sa ***P-LIST***).
De même le type d'un pointeur n'était pas direct, il fallait avoir accès au CAR du doublet pour savoir s'il s'agissait d'un élément terminal de type symbole atomique.

Pour résoudre ces 2 problèmes, les systèmes VLISP stockent les symboles en dehors de la zone allouée aux listes. Un symbole atomique est un pointeur sur une zone mémoire contenant les propriétés naturelles du symbole.

Notre modèle donne 8 propriétés naturelles à chaque symbole :

C-VAL (abréviation de Cell-value).
La valeur associée à cette propriété contient à tout moment la valeur VLISP associée à ce symbole considéré comme une variable. L'accès à cette propriété doit être très rapide, car c'est la propriété la plus utilisée (voir le chapitre 4).

P-LIST (abréviation de Property List).
La valeur associée à cette propriété est la liste des propriétés du symbole créées par l'utilisateur. Cette nouvelle liste de propriétés se trouve dans la zone liste. Elle permet de définir un nombre illimité (si ce n'est par la taille de la mémoire allouée aux listes) de nouvelles proriétés.

F-VAL (abréviation de Function Value).
La valeur associée à cette propriété contient à tout moment la définition de la fonction VLISP associée au symbole si elle existe. Cette propriété est décrite au chapitre 4.

F-TYPE (abréviation de Function Type).
La valeur de cette propriété est le type de la fonction contenue dans la F-val du symbole. Cette propriété est décrite au chapitre 4.

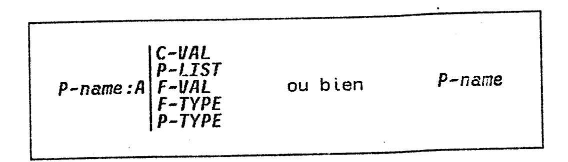
P-TYPE (abréviation de Print Type).
La valeur de cette propriété contient le type d'édition de la représentation externe du symbole. Cette propriété est décrite au chapitre 6.

A-LINK (abréviation de Atom Link).
La valeur de cette propriété contient le pointeur sur le symbole suivant de la zone symbole. Ce pointeur, indispensable du fait de la taille variable des propriétés naturelles d'un symbole permet en outre de gérer le hash-coding de la table des symboles. Cette propriété est décrite au chapitre 5.

P-LEN (abréviation de Print Length).
La valeur de cette propriété contient le nombre de caractéres du nom externe du symbole. Cette propriété est décrite au chapitre 6.

P-NAME (abréviation de Print Name).
La valeur de cette propriété contient le nom du symbole, i.e. les caractères de sa représentation externe. Cette propriété est décrite au chapitre 6.

Nous décrirons un symbole atomique par :



A la création d'un symbole, sa ***C-VAL*** contient la valeur spéciale *<*INDEFINIE*>* qui permet de tester les tentatives d'accès à une ***C-VAL*** non encore définie, sa P-list contient la valeur NIL, ainsi que les propriétés F-type et P-type.

La création de l'atome de nom FOO réalise donc :

```
      |<*INDEFINE*>
      |NIL
FOO:A |NIL
      |NIL
      |NIL
```

2.3 LE PROBLEME EPINEUX DES NOMBRES.

Toutes les machines actuelles sont orientées vers le traitement de données de type nombre. Or dans les systèmes LISP, c'est la représentation des nombres qui justement pose un problème. Pourquoi? Parceque dans la plupart des machines les objets de type adresse sont homomorphiques avec les objets de type nombre, ce qui permet de réaliser des opération arithmétiques (calculs et comparaisons) sur des adresses et de ne pas avoir d'instructions spéciales de manipulation d'adresses. En LISP il n'est donc pas possible de représenter les nombres uniquement par leur valeur. Enonçons le problème avec un exemple :

> la liste : **(1000 -30.26)**
>
> ne peut pas être représentée ainsi :
>
> $\{\ x{:}C\left|\begin{array}{l}1000\\ y\end{array}\right.,\quad y{:}C\left|\begin{array}{l}-30.26\\ NIL\end{array}\right.\ \}$

car LISP ne saurait pas si le CAR de cette liste est le nombre 1000 ou bien l'adresse 1000 qui peut être un pointeur sur un autre objet LISP. De même pour la seconde valeur -30.26 .

Plusieurs solutions sont envisageables. Nous les étudierons du point de vue :

- de la place occupée pour stocker les valeurs
- du temps d'exécution des opérations sur les valeurs
- des limitations de la représentation et des calculs
- de la rapidité du test de type

1) On peut utiliser des listes de chiffres pour représenter des nombres, ces chiffres étant considérés comme des symboles. C'est peut-être la solution la plus lispienne mais la moins efficace à la fois du point de vue occupation mémoire et temps d'exécution que du point de vue temps du test de type. Toutefois, commme cette représentation utilise les primitives d'accès et de construction des listes, il n'y a pas de limites quant à la taille des nombres et cela permet de faire des calculs arithmétiques en longeur variable. Cette représentation n'est plus jamais utilisée pour des raisons d'efficacité. Toutefois les packages d'arithmétiques à longeur variable (les ***Bignums***) utilisent cette méthode mais avec des listes de nombres dépendants du nombre de bits disponibles par mot-mémoire.

2) On utilise un doublet de liste, à la manière des symboles atomiques. Le CAR de ce doublet contient l'indicateur spécial *<"NOMBRE">* indiquant d'une part le type terminal de ce doublet et d'autre part que le CDR contient non pas un pointeur LISP mais une valeur numérique.

nombre	=	*x:C*	*<"NOMBRE">*
			valeur numérique

Cette représentation qui permet d'utiliser les possibilités de la machine quant aux calculs mais occupe deux fois plus de mémoire que nécessaire et ne permet pas un test direct du type du pointeur (ce test étant réalisé après accès au CAR). En revanche l'indicateur *<"NOMBRE">* peut servir à typer la valeur numérique et prendre les noms *<"NOMBRE ENTIER">, <"NOMBRE FLOTTANT">*.

3) Comme pour les symboles atomiques, il est également possible de stocker les valeurs numériques dans des zones mémoire différentes de celle des listes. Ces zones mémoires (une par type de nombre : entiers, flottants, flottants double précision ...), ne contiendront que des valeurs numériques.

Nous décrirons donc les nombres ainsi :

valeur:N|	ou	*valeur*	pour les nombres entiers
valeur:F|	ou	*valeur*	pour les nombres flottants

Cette représentation permet d'utiliser au mieux les possibilités de la machine, de n'occuper que l'espace strictement nécessaire et permet de réaliser le test de type par simple comparaison d'adresse. Cette méthode demande cependant la création de modules spécialisés d'allocation et de récupération de l'espace réservé aux nombres.

Nos réalisations utiliseront cette dernière méthode.

2.4 COMMENT DIFFERENCIER LES TYPES ?

Une des opérations de l'interprète les plus fréquentes sur les pointeurs LISP est le test de type. En effet, presque toujours avant de traiter un objet, il faut en déterminer le type, pour demander des conversions de type automatiques, pour provoquer des erreurs, voire pour arrêter le traitement.

Si la représentation des objets LISP n'utilise que des doublets de liste, les tests de type se résument à une consultation de la partie CAR de ce doublet qui contient dans le cas d'objets terminaux des indicateurs spéciaux. Ce mode de test oblige un accès à la mémoire et une suite de tests des indicateurs spéciaux (pour chaque type d'objets terminaux du système), ce qui est long et encombrant.

Si les valeurs des objets LISP sont rangées dans des zones propres à chaque type, le type d'un objet devient une fonction de son adresse. Dans cette organisation deux cas peuvent se produire :

1) la zone allouée à un type occupe un espace mémoire d'un seul tenant limité par les adresses ***Dz*** pour le début de la zone et ***Fz*** pour la fin de la zone.
   Le test de type se ramène à un test d'adresse :

   Dz < pointeur < Fz

2) la zone allouée à un type est en plusieurs morceaux. On suppose dans ce cas que la mémoire est partagée en un certain nombre de blocs de taille fixe. A chacun de ces blocs est assigné un type (statiquement à l'initialisation du système ou dynamiquement en cas de remplissage de l'espace alloué à un type) que le système gère en tenant à jour une table de types des blocs ***{Note 1}***. Le test de type consiste alors à déterminer le bloc dans lequel se trouve le pointeur puis à tester le type qui se trouve dans la table des types des blocs. Cette manipulation est dans la plupart des machines actuelles plus lente que la méthode vue précédemment car il est en général peu aisé d'isoler la partie de l'adresse qui fait office de numéro de bloc.

{Note 1} Une des peu nombreuses réalisations de ce type, en MACLISP est appelée BIBOP (acronyme de BIg Bag Of Pages) [STEELE 74, BAKER 77b].

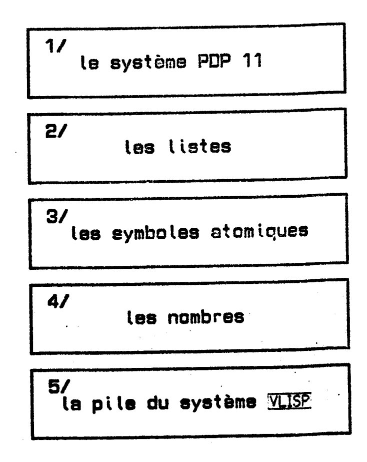
2.5 UNE INCARNATION : LE SYSTEME VLISP 11.

Voici à titre d'exemple d'implantation réelle les représentations des objets VLISP utilisés dans le système VLISP 11 *{Note 1}* fonctionnant sur ordinateur PDP-11 de Digital Equipment [DEC 75a] sous moniteur RT11 [DEC 78c].

Description sommaire de la machine PDP-11 :

- machine à mots de 16 bits
- mémoire adressable par octets ou par mots de 16 bits
- taille maximum de la mémoire 28k mots de 16 bits
- 8 registres banalisés de 16 bits (parmi lesquels se trouvent le compteur ordinal lui-même et le pointeur de pile)
- une pile en mémoire vive
- un puissant mode d'adressage

La mémoire est divisée en 6 zones de tailles fixes qui contiennent respectivement :



{Note 1} La réalisation de ce système est due à Patrick GREUSSAY [GREUSSAY 79b].

6/ l'interprète VLISP 11

La zone contenant l'interprète et la zone contenant le système PDP 11 contiennent des instructions machine. Les 4 autres zones contiennent des objets VLISP.

Toute référence à un objet VLISP est réalisée par un pointeur de 16 bits qui contient son adresse absolue.

2.5.1 Les listes

Un doublet de liste est représenté par 2 mots contigus dans la zone mémoire allouée aux listes. Le premier mot contient le pointeur CAR, le deuxième le pointeur CDR du doublet.

```
pointeur sur un doublet ->        [ pointeur CAR ]  (1 mot)
                                  [ pointeur CDR ]  (1 mot)
```

L'accès au pointeur CAR est réalisé par l'instruction :

```
MOV (source),destination
```

L'accès au pointeur CDR est réalisée par l'intruction :

```
MOV +2(source),destination
```

Il est également possible d'utiliser les modes d'adressages auto-incrément et auto-décrément de la machine PDP11 et réaliser l'accès au pointeur CAR puis l'accès au pointeur CDR en 2 instructions de 1 mot chacunes :

```
MOV (source)+,destination-CAR
MOV (source) ,destination-CDR
```

2.5.2 Les symboles atomiques

La référence à un symbole atomique est un pointeur vers un descritif stocké dans la zone mémoire allouée aux symboles atomiques. Ce descriptif de longeur variable contient les propriétés naturelles du symbole. Ces propriétés sont rangées de la manière suivante :

```
référence au symbole →  [ C-valeur du symbole ]  (1 mot)
                        [ P-liste  du symbole ]  (1 mot)
                        [ F-valeur du symbole ]  (1 mot)
                        [ F-type ]               (1 octet)
                        [ P-type ]               (1 octet)
                        [ A-link   du symbole ]  (1 mot)
                        [ P-len  ]               (1 octet)
                        [ P-name ......       ]  (n octets)
```

La zone mémoire nécessaire pour représenter un symbole atomique est donc égale à 11+n octets (n étant le nombre d'octets du P-name i.e. son *P-LEN*). Les références aux différentes propriétés naturelles sont réalisées directement en utilisant l'adressage indexé du PDP 11.

2.5.3 Les nombres

Il existe trois types de nombres dans le système VLISP 11.
- les petits nombres entiers positifs (de 0 à 2000)
- les autres nombres entiers (représentés sur 32 bits)
- les nombres flottants (représentés sur 32 bits également).

1) les petits nombres entiers positifs sont représentés par un pointeur sur la zone interprète elle-même.

2) les autres nombres entiers sont rangés à raison de 2 mots de 16 bits par valeur dans une partie de la zone réservée aux nombres.

3) et les nombres flottants, également rangés à raison de 2 mots de 16 bits par valeur, dans l'autre partie de la zone réservée aux nombres.

Les 4 zones contenant les objets VLISP (listes, nombres, symboles et pile) sont gérées dynamiquement.

Enfin les zones occupant des espaces contigus et fixes de la mémoire, le test de type peut être réalisé par simple comparaisons d'adresses.

CHAPITRE 3
LA MACHINE V C M C 2

Ce chapitre décrit la machine VCMC2 utilisée dans cette étude.
La conception et l'utilisation de la machine VCMC2 ont été conduites par les idées suivantes :

1) VCMC2 est une machine.

La description de l'implantation de notre modèle dans cette machine prend en compte les limitations inhérentes à l'utilisation d'une machine :
- les ressources (de type mémoires ou registres) sont limitées
- toute manipulation d'objets doit être explicitée au moyen d'un jeu d'instruction fixe.

2) VCMC2 est une machine virtuelle.

Elle permet de se libérer des contraintes d'une machine spécifique, notamment au niveau de la représentation interne des objets.

3) VCMC2 est une machine référentielle.

La description du modèle est portable (les trois systèmes ont été construit très rapidement en quelques semaines) et concise (la taille de l'interprète et des 160 fonctions standard est de l'ordre de 2 k mots).

4) VCMC2 est une machine prototype.

Elle sert d'étude préliminaire à la réalisation d'une machine VCMC2 réelle contruite autour d'un micro-processeur bipolaire à tranches tel le Am2900 [AMD 78a]

3.1 ORGANISATION DE LA MACHINE.

La machine VCMC2 est composée d'une mémoire, de registres et d'une Unité Arithmétique et Logique.

3.1.1 La mémoire

La mémoire de la machine VCMC2 est une mémoire à accès aléatoire. La taille de ses mots n'est pas fixée et dépend des incarnations particulières de la machine sur des matériels existants. Un mot VCMC2 doit pouvoir contenir une adresse. De la taille d'un mot va dépendre donc la taille maximale de la mémoire (son espace-adresse) *{Note 1}*.

Cette mémoire est découpée en plusieurs zones. Chacune de ces zones contient un type de donnée fixé.

On distingue les zones suivantes :

1- la zone mémoire contenant les programmes (l'interprète, les fonctions d'entrée-sortie, les mémoires de travail ...). Cette zone n'est pas allouée dynamiquement. On appelle cette zone la zone code.

2- la pile. Cette zone mémoire est géré automatiquement par la machine au moyen d'instructions spécifiques faisant intervenir un pointeur spécial, le pointeur de pile. Cette pile peut être conçue comme un processeur externe avec sa propre mémoire à la manière de la pile de la machine LISP NK3 [NAGAO 79].

3- la zone mémoire contenant les listes. Cette zone manipulable par des intructions spécifiques de la machine est gérée dynamiquement.

4- la zone mémoire contenant les nombres. Comme pour la zone précédente, la gestion de cette zone est automatique et dynamique.

5- la zone mémoire contenant les symboles atomiques, qui est gérée elle aussi dynamiquement.

La notion de zones mémoire peut être assimilée à la notion de SEGMENTs mémoire tels qu'ils sont apparus dans les nouveaux micro-processeurs à mots de 16 bits comme le Intel/8086 [INTEL 79] ou le Z8000 [AMD 79].

{Note 1} la plupart des machines avaient jusqu'à maintenant des adresses sur 16 bits qui ne permettaient d'adresser que 64 k mots, ce qui est très insuffisant pour des machines VLISP. La tendance actuelle (avec les micro-processeurs à mots de 16 bits) est à l'augmentation des tailles des adresse : 20 bits pour le Intel 8086 [INTEL 79] permettant ainsi d'adresser 1 M-octets, 23 bits pour le Zilog 8000 [AMD 79] ou le Motorola 68000 [MOTOROLA 79]) permettant d'adresser 8 M-mots.

3.1.2 Les registres

Pour des raisons de rapidité d'exécution, la machine VCMC2 dispose de mémoires rapides (ou registres). Le nombre des registres est limité à 7 dans notre modèle.
Trois d'entre eux sont affectés à des taches particulières (PC, SP, IX) et un nombre de 4 registres de travail a été retenu au vu des statistiques d'utilisation des registres de travail *{Note 1}*. En outre le nombre limité de registres facilite les implémentations du modèle sur les machines existantes comme le PDP11 (à 8 registres), le 8080 (à 5 registres) *{Note 2}*, le PDP10, Z8000, Am2900 à 16 registres.

Il y a 3 registres spécialisés, utilisés par la machine, et 4 registres de travail banalisés, utilisés par l'interprète.

1- PC : ce registre est le compteur ordinal de la machine VCMC2 et contient donc, à tout moment, l'adresse de l'instruction suivante à exécuter par la machine. PC fait office de pointeur sur le programme.

2- SP : ce registre est le pointeur de pile. Il contient à tout moment l'adresse du dernier mot empilé et est actualisé à chaque modification de cette pile.

3- IX : ce registre est le seul registre d'index de la machine et est utilisé pour pointer sur les mémoires de travail de la zone code.

4- A1 : est le premier registre de travail (ou accumulateur 1)

5- A2 : est le deuxième registre de travail (ou accumulateur 2)

6- A3 : est le troisième registre de travail (ou accumulateur 3)

7- A4 : est le quatrième registre de travail (ou accumulateur 4)

{Note 1} les statistiques données au paragraphe 3.3.2. montrent que le premier registre de travail est utilisé dynamiquement pour 37% des opérandes, le second pour 19% des opérandes, le troisième pour 7% et le quatrième pour moins de 3%. L'ajout de registres de travail supplémentaires n'est donc pas nécessaire.

{Note 2} le 8030 ne possède que 5 registres pouvant contenir des pointeurs : PC, SP, HL, DE et BC. Les registres manquants sont remplacés par des mots en mémoire ce qui permet d'utiliser le modèle mais fait perdre de l'efficacité.

3.1.3 L'unité arithmétique et logique (U.A.L.)

VCMC2 doit disposer d'une UAL universelle permettant de réaliser aussi bien des manipulations de pointeurs, que des tests de type ou encore des opérations arithmétiques. Les UALs bipolaires de type Am2903 [AMD 78a] sont parfaitement suffisantes pour la partie logique et l'unité arithmétique Am9511 [AMD 78c] (quoique un peu lente : 54-368 cycles pour l'addition flottante 32 bits) permet des calculs arithmétiques sur des nombres entiers représentés sur 16 ou 32 bits et sur des nombres flottants représentés sur 32 bits.

3.2 LES OBJETS TRAITES ET LES OPERANDES.

La machine VCMC2 va traiter des pointeurs. Un pointeur est l'adresse d'un objet VLISP à laquelle est associé un type. Ce type est la zone mémoire dans laquelle il est contenu. Tout accès à la valeur d'un objet VLISP (ou aux différentes valeurs si l'objet est un symbole atomique) s'opère par indirection au travers de ce pointeur, dans la zone mémoire correspondante, au moyen d'instructions propres à chaque valeur.

La machine VCMC2 possède 4 types de pointeurs qui sont :
- le type symbole atomique
- le type nombre
- le type liste
- le type objet binaire (qui englobe tout ce qui n'est pas objet VLISP, i.e. la zone pile, et la zone code qui contient les instructions et les mémoires de travail de l'interprète).

Ces types ont été définis sur la base d'une gestion (allocation et récupération) spécifique de chacune des zones mémoire. La taille des pointeurs n'est pas déterminée par la machine VCMC2 mais par chacune de ses incarnations particulières. De même le nombre de types disponibles peut être augmenté pour tenir compte du matériel utilisé.

Voici les types disponibles dans une machine à 8 types :

0	000	pointeur sur un symbole atomique
1	001	pointeur sur un nombre entier
2	010	pointeur sur un nombre réel
3	011	pointeur sur une chaîne de caractères
4	100	pointeur sur un doublet de liste
5	101	pointeur sur une liste compactée
6	110	pointeur sur un élément de la pile
7	111	pointeur sur une instruction

L'exécution de la plupart des intructions s'opère sur des opérandes de type défini (par exemple l'addition s'opére entre 2 opérandes de type nombre, l'instruction CAR sur un opérande de type liste).
La machine VCMC2 opére un contrôle de validité de type à chaque exécution d'une instruction et provoque une erreur avec interruption dans la machine, en cas de mauvaise utilisation de pointeur en lecture ou en écriture.

3.3 LES CHAMPS D'UNE INSTRUCTION ET LEUR DECODAGE.

Une instruction est la description d'une action à réaliser par la machine VCMC2. Ces instructions sont stockées dans la zone mémoire réservée à cet effet, la zone code. Chaque instruction est décrite au moyen d'un à quatre mots de la machine, la machine VCMC2 utilise donc des instructions de longeur variable. Cette architecture de machine que l'on rencontre dans les ordinateurs de type PDP/11 ou Intel/8086, permet d'utiliser des opérandes explicites et de gérer au mieux la place mémoire i.e. en minimisant l'espace occupé par les instructions.

Le 8086 [INTEL 79] a des instructions codées sur 8 bits et possède 3 formats d'instruction de saut inconditionnel :

1) Le saut relatif.
le mot suivant l'instruction contient un déplacement qui ajouté au PC livre l'adresse de branchement. Cette instruction est codée sur 2 mots. Un mot contient l'instruction et l'autre le déplacement :

1 1 1 0 1 0 1 1	*Saut relatif*
x x x x x x x x	*Déplacement*

2) Le saut direct dans le segment code.
Les 2 mots suivants contiennent l'adresse de saut à l'intérieur du segment code sur 16 bits. Cette instruction est codée sur 3 mots. Un mot contient l'instruction et les 2 mots suivants l'adresse :

1 1 1 0 1 0 0 1	*Saut dans le segment*
x x x x x x x x	*Adresse basse*
x x x x x x x x	*Adresse haute*

3) Le saut intersegment.
Les 2 mots suivants contiennent l'adresse de saut à l'intérieur d'un segment dont l'adresse est donnée également dans les 2 mots suivants le déplacement. Cette instruction est codée sur 5 mots. Un mot contient l'instruction, 2 mots le déplacement et 2 mots l'adresse du segment.

1 1 1 0 1 0 1 0	***Saut intersegment***
x x x x x x x x	***Adresse basse***
x x x x x x x x	***Adresse haute***
x x x x x x x x	***Adresse basse segment***
x x x x x x x x	***Adresse haute segment***

Cette disposition permet donc de minimiser la taille des programmes, en occupant au mieux l'espace programme.

Dans la machine VCMC2 le premier mot de chaque instruction possède un format unique pour toutes les intructions, ce qui permet un décodage simplifié et rapide en cas de réalisation effective.

Ce premier mot est découpé en 4 champs :
- le champ code instruction
- le champ premier opérande (opérande source)
- le champ second opérande (opérande destination)
- et le champ CONTINUATION

Ces différents champs vont être décrits fonctionnellement.

3.3.1 <u>Le champ code instruction</u>

Le champ code instruction contient l'action à réaliser sur les opérandes. Ce champ est représenté par le nom mnémonique de l'instruction. Seul ce champ est obligatoire pour toutes les instructions. Les différentes instructions sont décrites au paragraphe 3.4 ainsi que les tablaux d'apparition statique et dynamique des différentes instructions.

3.3.2 Les champs opérandes

Chaque instruction peut contenir la spécification d'1 ou de 2 opérandes qui vont intervenir durant son exécution. Le premier opérande s'appelle opérande source et le second opérande destination, par analogie avec l'intruction de transfert. Cette instruction qui se nomme MOVE, possède 2 opérandes. Le premier est l'émetteur (la source) et le second le récepteur (la destination). L'instruction transfère le contenu de l'émetteur dans le récepteur.

La machine VCMC2 possède 8 opérandes standard :

1) la constante NIL.

Opérande	NIL

Cet opérande est utilisé pour spécifier la constante NIL (sans utiliser les instructions spéciales de manipulation de symboles atomiques). Il ne doit jamais apparaître en position opérande destination, sous peine de déclencher une erreur de la machine VCMC2. Il n'est pas possible en effet de modifier une des propriétés naturelles de la constante NIL car la machine les utilise fréquemment pour ses besoins propres et ne teste pas leur validité pour des raisons d'efficacité.

2) l'un des 4 registres généraux (accumulateurs) de la machine.

Opérandes	A1, A2, A3 ou A4

3) le sommet de la pile.

Opérande	TST

Cet opérande permet d'utiliser la pile. Employé comme opérande source, il réalise une opération de dépilage, et employé comme opérande destination, il réalise une opération d'empilage. Dans ces deux cas, le pointeur de pile (SP) est actualisé pour pointer sur le dernier mot empilé et dans ces deux cas la pile est modifiée. En cas de débordement de la pile ou de vidage de celle-ci, une erreur de la machine VCMC2 apparaît.

4) un pointeur explicite qui se trouve dans le mot mémoire suivant l'instruction. Ce pointeur peut-être un pointeur sur un objet VLISP ou bien un pointeur sur la zone mémoire contenant les programmes (ces pointeurs sont représentés au moyen d'étiquettes dans la zone code). Cet opérande s'appelle opérande immédiat.

Opérande | 'objet-VLISP ou (adresse-mémoire)

5) un pointeur dont l'adresse se trouve dans le mot mémoire suivant l'intruction. Cet opérande permet d'accéder directement à des mots mémoire de la zone code dont on connait l'adresse. Cet opérande s'appelle opérande direct.

Opérande | (@ adresse-memoire)

Voici les fréquences d'utilisation des 8 types d'opérandes de la machine VCMC2. Les fréquences statiques ont été calculées en analysant tout le code VCMC2 des appendices **C**, **E** et **F**. Les fréquences dynamiques ont été calculées en analysant le test de l'appendice **G**.

<u>Occurrences statiques des opérandes</u>

Nombre d'opérandes visités : 2640
Nombre d'instructions : 1320

1	NULL	= 476	18.03 %	36.06 %
2	A1	= 650	24.62 %	49.24 %
3	A2	= 346	13.11 %	26.21 %
4	A3	= 213	8.07 %	16.14 %
5	A4	= 160	6.06 %	12.12 %
6	TST	= 343	12.99 %	25.98 %
7	IMMEDIAT	= 288	10.91 %	21.82 %
8	DIRECT	= 164	6.21 %	12.42 %

Occurrences dynamiques des opérandes

Nb d'opérandes évalués : 148113
Nb d'instructions exécutées : 80202

1	NULL	= 6575	4.44 %	8.2 %
2	A1	= 50714	34.24 %	63.23 %
3	A2	= 26711	18.03 %	33.3 %
4	A3	= 9543	6.44 %	11.9 %
5	A4	= 5486	3.7 %	6.84 %
6	TST	= 18850	12.73 %	23.5 %
7	IMMEDIAT	= 18357	12.39 %	22.89 %
8	DIRECT	= 11877	8.02 %	14.81 %

Ces données indiquent que :

- l'utilisation intensive des registres généraux : plus de 50% en statique et plus de 60% en dynamique. Ces chiffres montrent bien l'utilité d'un petit nombre de registres rapides qui font gagner 60% des accès mémoire réalisés par la machine pour accéder aux opérandes.

- la fréquence d'utilisation décroissante des différents registres de travail (A1, A2, A3, A4) pour arriver jusqu'à 4% d'utilisation dynamique pour A4, ce qui montre que le nombre de 4 registres de travail est bien adapté et que le rajout d'autres registres n'améliorerait pas sensiblement les performances du modèle.

3.3.3 Le champ continuation

Chaque instruction possède un quatrième champ, le champ continuation, qui spécifie l'adresse de l'instruction suivante à exécuter *{Note 1 et 2}*.

L'introduction de ce dernier champ permet :

1) de réduire la place occupée par les programmes. Toute instruction possédant un champ continuation, les instructions de contrôle de séquence deviennent inutiles.

2) de gérer les ruptures de séquence conditionnelles en les réalisant par inhibition ou validation de l'interprétation du champ continuation.

3) de faciliter la réalisation de la micro-machine qui pourra utiliser directement ce champ pour commander le séquenceur (à la manière du micro-séquenceur Am2911 de la famille 2900 [AMD 78b]).

Pour réduire l'espace occupé par ce champ, l'adresse de l'instruction suivante peut être soit implicite, par utilisation du registre PC ou de la pile, soit explicite et dans ce cas il faut pouvoir spécifier une nouvelle adresse.

Nous avons donc choisi quatre continuations standard dans la machine VCMC2 :

1) la continuation NOP.
Elle indique la séquentialité d'exécution, la prochaine instruction à exécuter se trouve à la suite, il n'y a pas de rupture de séquence.

Continuation |	[NOP]	ou bien omission du champ

2) la continuation JUMP.
Elle indique une rupture de séquence (un branchement). L'adresse de l'intruction suivante à exécuter se trouve dans le mot mémoire

---

{Note 1} ce champ qui est présent dans les micro-machines se retrouve même dans des langages très évolués, par exemple le GOTO FIELD de SNOBOL 4 [GRISWOLD 71].

{Note 2} La notion de continuation évoquée ici n'est pas totalement étrangère à son emploi en sémantique dénotationnelle [ROBINET 79] mais s'en distingue par son orientation résolument opérationnelle.

qui suit l'intruction.

Continuation	[JUMP (adresse-memoire)]

3) la continuation CALL.
Elle indique un appel de sous-programme. L'adresse du sous-programme à exécuter se trouve dans le mot mémoire suivant l'instruction (d'une manière identique à la continuation JUMP). De plus avant d'exécuter ce branchement, l'adresse de l'instruction qui devait être exécutée en séquence est empilée dans la pile pointée par SP.

Continuation :	[CALL (adresse-mémoire)])

4) la continuation RETURN.
Elle indique un retour de sous-programme. L'adresse de l'instruction suivante à exécuter se trouve dans le sommet de la pile. Après dépilage de cette adresse, le pointeur de pile (SP) est mis à jour.

Continuation	[RETURN]

Les tableaux suivants montrent la fréquence d'apparition des champs continuations dans la machine VCMC2, avec les mêmes tests que pour l'analyse des champs opérandes du paragraphe précédent.

Occurrences statiques des continuations

Nombre de champs continuation : 664
Nombre d'instructions : 1320

1	JUMP	= 311	46.84 %	23.56 %
2	CALL	= 163	24.55 %	12.35 %
3	RETURN	= 190	28.61 %	14.39 %

<u>Occurrences dynamiques des continuations</u>

Nb de continuations : 25058
Nb d'instructions exécutées : 80202

1	JUMP	=	14708	58.7 %	18.34 %
2	CALL	=	3552	14.18 %	4.43 %
3	RETURN	=	6798	27.13 %	8.48 %

Ces tableaux nous montre l'utilisation intensive de ces champs, de manière statique (plus de 50% des instructions ont un champ continuation différent de NOP, donc utilisé) et de manière dynamique (plus de 30% des instructions exécutées possèdent un champ continuation).

Le gain en occupation mémoire apporté par son utilisation est très important. En effet si le champ continuation d'une instruction est codé sur ***<bcont>*** bits, et une instruction sur ***<binst>*** bits, chaque continuation faisant gagner un mot mémoire (qui contiendrait l'instruction de branchement si la machine ne possède pas ce champ) le gain en mots obtenu par l'utilisation de ***<ncont>*** champs continuation dans un ensemble de ***<ninst>*** instructions est de :

gain = ((<ncont> * <binst>) - (<ninst> * <bcont>)) / <binst>

Dans la machine VCMC2 à pointeurs de 16 bits, les champs continuation sont codés sur 2 bits. Le gain, en mots, dans le cas de l'étude statique (1283 instructions occupant 2061 mots et 648 continuations) est de :

((664 * 16)-(1320 * 2)) / 16

i.e. 499 mots.

donc sur 2125 mots un gain de 24%

3.3.4 Représentation d'une instruction

Chaque instruction est codée dans un mot de la mémoire, ce mot pouvant être suivi de un à trois pointeurs (un pointeur décrivant l'opérande source, un pointeur décrivant l'opérande destination et un pointeur décrivant la continuation). Il y a donc 4 formats possibles d'instructions : les instructions occupant 1 mot, 2 mots, 3 mots et 4 mots.

Les tableaux suivants montrent la fréquence d'apparition, statique et dynamique, de chacun de ces formats, en utilisant les mêmes tests que pour les champs précédents :

Occurrences statiques des formats

Nombre d'instructions : 1320
Nombre de mots : 2125

1	1MOT	= 650	49.24	%	0	%
2	2MOTS	= 550	41.67	%	0	%
3	3MOTS	= 105	7.95	%	0	%
4	4MOTS	= 15	1.14	%	0	%

Occurrences dynamiques des formats

Nb d'instructions exécutées : 80202
Nb de mots des instructions : 128588

1	1MOT	= 40328	50.28	%	0	%
2	2MOTS	= 31376	39.12	%	0	%
3	3MOTS	= 8484	10.58	%	0	%
4	4MOTS	= 14	0	%	0	%

Ces tableaux montrent que la souplesse d'utilisation obtenue grâce aux instructions de longeur variable ne grève pas l'occupation de la mémoire, en effet la moyenne d'occupation par instruction est de 1,5 mots.

Voici la disposition des champs des instructions dans une machine à mots de 16 bits (chaque x représente 1 bit) *{Note 1}* :

code instruction	opérande source	opérande destination	champs continuation
x x x x x x x x	x x x	x x x	x x

Dans cette étude, les instructions VCMC2 sont représentées en utilisant un <u>langage d'assemblage</u>. Une instruction est décrite par une ligne qui contient :

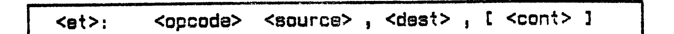
1) une étiquette optionnelle **<et>**: qui est un nom symbolique repérant l'instruction.

2) le nom mnémonique de l'instruction **<opcode>**. Ces noms ainsi que leurs actions sont décrites au paragraphe 3.4.

3) les arguments de l'instruction, i.e. les opérandes **<source> <dest>** et continuation **<cont>**, séparés entre eux par des virgules.

4) le champ continuation est encadré de crochets carrés **[<cont>]**

5) les commentaires sont annoncés par le caractère ; (point-virgule) et se terminent à la fin de chaque ligne.



{Note 1} cette taille qui réduit considérablement l'espace adresse à été conservée pour pouvoir implanter cette machine sur les machines Intel 8080 et PDP 11 qui possèdent des adresses sur 16 bits.

3.3.5 Evaluation d'une instruction

L'évaluation d'une instruction se déroule en 4 phases :

1) Recherche de l'adresse de l'opérande source.

2) Recherche de l'adresse de l'opérande destination.

3) Exécution de l'instruction. Toutes les instructions, en plus d'une action spécifique, positionnent un indicateur, le code-condition, CC. Cet indicateur peut avoir 2 valeurs : NIL ou T et est positionné en fonction du résulat de l'instruction.

4) Si le CC est égal à T (est vrai), le champ continuation est décodé puis exécuté. Si le CC est égal à NIL (est faux), le champ continuation n'est pas décodé et la continuation NOP est systématiquement exécutée (aprés avoir sauté l'opérande du champ continuation dans le cas d'une continuation de type JUMP ou CALL non exécutée).

   L'évaluation des différents champs se déroule dans un ordre précis : opérande source, puis opérande destination enfin champ continuation, ce qui permet de lever l'ambiguité dans la position des opérandes dans le cas où plusieurs arguments sont situés sous l'instruction.

   Par exemple l'instruction :

```
<opcode>    'FOO,'BAR,[JUMP (RE)])
```

est rangée sur 4 mots dans la zone code de la manière suivante :

```
<opcode>
'FOO
'BAR
(RE)
```

La machine considérera 'FOO comme étant le premier opérande, 'BAR comme second opérande et (RE) comme adresse du champ continuation.

3.4 DESCRIPTION DES INSTRUCTIONS.

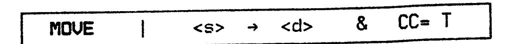
Ce paragraphe contient la description de toutes les intructions VCMC2 et l'appendice A contient une description résumée de notre machine destinée à faciliter la lecture des différents segments de programmes qui suivront.

Voici les notations que nous utiliserons pour décrire les instructions :

<s>	l'opérande source
<d>	l'opérande destination
→	est transféré dans
↔	est échangé avec
CC=	la nouvelle valeur du code condition.
CVAL	l'attribut CVAL d'un symbole atomique
PLIST	l'attribut PLIST d'un symbole atomique
FVAL	l'attribut FVAL d'un symbole atomique
FTYP	l'attribut FTYP d'un symbole atomique
PTYP	l'attribut PTYP d'un symbole atomique
+	l'addition
-	la soustraction
*	la multiplication
/	la division
\	le reste de la division
ou	le ou inclusif logique
et	le et logique
oux	le ou exclusif logique
CAR	la partie CAR d'un doublet
CDR	la partie CDR d'un doublet
(x . y)	un doublet de liste, x est la partie CAR y est la partie CDR
SP	le registre pointeur de pile
(SP)	le contenu du sommet de pile
IX	le registre d'index
x[y]	un opérande indexé i.e. un opérande dont l'adresse est égale à : x + y

3.4.1 Le transfert

L'instruction **MOVE** transfère l'opérande source dans l'opérande destination.



Cette instruction est la plus utilisée : du fait de la puissance des opérandes, elle permet en effet tous les transferts :

registre ou mémoire ou pile	→ vers ←	registre ou mémoire ou pile

3.4.2 Manipulation des symboles atomiques

Les 10 instructions qui suivent vont permettre d'accéder aux propriétés naturelles *C-VAL, P-LIST, F-VAL, F-TYPE et P-TYPE* des symboles atomiques. Il existe 2 instructions pour chacune des propriétés, une instruction de lecture et une instructions d'écriture de la propriété. Elles permettent des transferts de type :

registre ou mémoire ou pile	→ vers ←	la zone des symboles atomiques

CVAL	(CVAL <s>)	→	<d>	& CC= T
SCVAL	<s>	→	(CVAL <d>)	& CC= T
PLIST	(PLIST <s>)	→	<d>	& CC= T
SPLIST	<s>	→	(PLIST <d>)	& CC= T
FVAL	(FVAL <s>)	→	<d>	& CC= T
SFVAL	<s>	→	(FVAL <d>)	& CC= T
FTYP	(FTYP <s>)	→	<d>	& CC= T
SFTYP	<s>	→	(FTYP <d>)	& CC= T
PTYP	(PTYP <s>)	→	<d>	& CC= T
SPTYP	<s>	→	(PTYP <d>)	& CC= T

NB : Le préfixe S (pour SET) devant les instructions indique un accès en écriture de la propriété.

3.4.3 Manipulation des nombres

Ces instructions réalisent les opérations arithmétiques. Si les opérandes ne sont pas de type nombre ou si les instructions provoquent des exceptions arithmétiques (débordement de capacité, division par 0), une erreur *{Note 1}* de la machine VCMC2 se produit. La machine se charge de la gestion de la zone allouée aux nombres et en particulier du stockage de la valeur du résultat.

ADD	<s> + <d>	→	<d>	&	CC= T
SUB	<s> - <d>	→	<d>	&	CC= T
MUL	<s> * <d>	→	<d>	&	CC= T
QUO	<s> / <d>	→	<d>	&	CC= T
REM	<s> \ <d>	→	<d>	&	CC= T
LOGOR	<s> ou <d>	→	<d>	&	CC= T
LOGAND	<s> et <d>	→	<d>	&	CC= T
LOGXOR	<s> oux <d>	→	<d>	&	CC= T

{Note 1} en cas d'erreur le programme est automatiquement dérouté (au moyen d'un TRAP) vers un module spécialisé de traitement d'exception.

3.4.4 Manipulation des pointeurs sur les listes

Ce groupe d'instructions permet d'accéder à la zone des doublets de liste. Cette zone est gérée automatiquement par la machine VCMC2. En particulier le récupérateur de cette zone est partie intégrante de la machine VCMC2. Si le récupérateur ne peut plus fonctionner (si la zone est pleine) une erreur VCMC2 se déclenche.

	registre	→	la zone
ou	mémoire	vers	des doublets
ou	pile	←	de liste

CAR	(CAR <s>)	→	<d>	& CC= T
CDR	(CDR <s>)	→	<d>	& CC= T
SCAR	<s>	→	(CAR <d>)	& CC= T
SCDR	<s>	→	(CDR <d>)	& CC= T
CONS	(<s> . <d>)	→	<d>	& CC= T
XCONS	(<d> . <a>)	→	<d>	& CC= T

Les intructions **SCAR** et **SCDR** correspondent aux primitives de manipulation de liste RPLACA et RPLACD.

Du fait de la non-symétrie des opérandes (NIL ne peut pas se trouver en position destination, TST empile ou dépile en fonction de sa position) il existe 2 instructions symétriques de création de doublet de liste : **CONS** et **XCONS** qui permettent ainsi toutes les constructions.

```
CONS  NIL,A1 → (A1 .  NIL) i.e.  (A1)
CONS  A1,NIL est illégal
XCONS NIL,A1 → (NIL .  A1)
```

3.4.5 Utilisation de la Pile

Cette famille d'instructions permet d'utiliser la pile et le pointeur de pile SP, indépendamment de l'opérande TST.

STACK	SP	→	<d>	& CC= T
SSTACK	<s>	→	SP	& CC= T
TOPST	(SP)	→	<d>	& CC= T
XTOPST	(SP)	↔	<s>	& CC= T

Les 2 instructions **STACK** et **SSTACK** permettent d'avoir accès en lecture et en écriture au registre pointeur de pile lui-même.
L'instruction **TOPST** permet d'accéder au sommet de pile sans modifier la pile ni le pointeur de pile, cette instruction est donc utilisée pour consultation non-destructive du sommet de la pile.
Enfin l'instruction la plus puissante de manipulation de pile et **XTOPST** qui échange le sommet de la pile avec l'opérande de l'instruction sans modifier le pointeur de pile *{Note 1}*.

3.4.6 Utilisation du registre d'Index IX

Ces instructions vont utiliser le registre d'index soit directement, soit pour calculer la véritable adresse de l'opérande (dans ce cas, la valeur du registre IX est ajoutée à l'opérande, cette somme produisant l'adresse de l'opérande à utiliser).

INDEX	IX	→	<d>	& CC= T
SINDEX	<s>	→	IX	& CC= T
MOVEX	<s>	→	<d>[IX]	& CC= T
XMOVE	<s>[IX]	→	<d>	& CC= T

{Note 1} une version affaiblie de cette instruction existe même dans le micro-processeur 8080 [INTEL 77a] dont l'instruction XTHL réalise l'échange du sommet de la pile et du registre HL. Cette instruction de 18 cycles n'est codée que sur 1 octet!

3.4.7 Les instructions de contrôle

Ces intructions sont très peu nombreuses du fait de l'utilisation du champ continuation des instructions. Seules 2 instructions d'aiguillage ont été introduites :

JUMPX		<s> [<d>]	→ PC	&	CC= NIL
DISPT	si (LITATOM <d>)	<s> [0]	→ PC	&	CC= NIL
	si (NUMBP <d>)	<s> [1]	→ PC	&	CC= NIL
	si (LISTP <d>)	<s> [2]	→ PC	&	CC= NIL

L'instruction **JUMPX** réalise un aiguillage au travers une table d'étiquettes dont l'adresse est donnée en opérande source. L'indice dans cette table est l'opérande destination (qui bien qu'en position destination est considéré par VCMC2 comme un opérande source).

L'instruction **DISPT** réalise également un aiguillage. L'indice de la table d'étiquettes est fourni par le type de l'opérande destination. Cette instruction permet de réaliser un test de type en un temps indépendant du nombre de types.

Les types suivants sont disponibles :

0	type symbole atomique
1	type nombre
2	type liste

3.4.8 Les tests de type

Ces instructions permettent de tester explicitement le type d'un pointeur (symbole atomique, nombre ou liste). Elles ne possèdent qu'un opérande, et seul le code condition est affecté par ces instructions.

TATOM	CC= (LITATOM <s>)
FATOM	CC= (NOT (LITATOM <s>))
TNUMB	CC= (NUMBP <s>)
FNUMB	CC= (NOT (NUMPB <s>))
TLIST	CC= (LISTP <s>)
FLIST	CC= (NOT (LISTP <s>))

N.B. : les préfixes T et F indiquent respectivement vrai (T) ou faux (F).

3.4.9 Les tests d'égalité de pointeurs

Ces deux instructions testent l'égalité ou l'inégalité de 2 pointeurs. Deux pointeurs sont dits égaux, s'ils possèdent la même adresse et le même type, donc s'ils pointent sur le même objet physique.

EQ	CC= (<s> = <d>)
NEQ	CC= (<s> ≠ <d>)

Ces instructions correspondent aux prédicats VLISP EQ et NEQ.

3.4.10 Les comparaisons arithmétiques

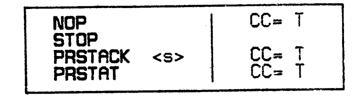
Elles sont utilisées pour comparer les valeurs des nombres et positionnent le code-condition.

GE		CC= (GE <s> <d>)
GT		CC= (GT <s> <d>)
LT		CC= (LT <s> <d>)
LE		CC= (LE <s> <d>)
SUBTZ	<d> - <s> → <d> &	CC= (ZEROP <d>)
SUBFZ	<d> - <s> → <d> &	CC= (NOT (ZEROP <d>))

Les 2 dernières instructions **SUBTZ** et **SUBFZ** sont utilisées pour réaliser commodément des boucles.

3.4.11 Les instructions spéciales

Il existe quelques instructions qui n'ont pas d'effet sur le contenu des registres ou des mémoires. Ces intructions réalisent des effets de bord divers et positionnent toujours le code-condition à T.



NOP	CC= T
STOP	
PRSTACK <s>	CC= T
PRSTAT	CC= T

NOP ne fait rien, mais permet de décoder le champ continuation.

STOP arrête la machine VCMC2.

PRINSTACK imprime le contenu des <s> derniers mots de la pile. Cette instruction permet de connaître l'état de la pile dynamiquement et réalise des contrôles dynamiques.

PRSTAT imprime le nombre d'instructions exécutées, le nombre de CONS réalisés et la taille maximum de la pile depuis le dernier PRSTAT. Cette instruction permet d'obtenir des statistiques incrémentales.

3.4.12 Les instructions d'entrée/sortie

Il existe 2 familles d'instructions. Les instructions d'entrée/sortie complexes, qui travaillent au niveau des expressions VLISP elles-mêmes et sont utilisées dans l'interprète, et les instructions de très bas niveau, manipulant des caractères, utilisées dans l'écriture des modules d'entrée/sortie. Ces dernières instructions seront vues dans les chapitres 5 pour les instructions d'entrée et 6 pour les instructions de sortie.

Voici les instructions d'entrée/sortie complexes :

READ	entrée → <d>	CC= T
PRINI	<s> → sortie	CC= T
TERPRI	saut de ligne	CC= T

READ lit une expression VLISP et charge sa valeur dans l'opérande <d>

PRINI imprime la valeur de l'opérande <s>

TERPRI provoque un saut de ligne en sortie.

3.5 LES PSEUDO-INSTRUCTIONS.

3.5.1 Les MACRO

En plus de ces instructions de base, VCMC2 permet d'utiliser des MACRO ce qui facilite l'écriture et la lecture des programmes.

Il existe 4 MACRO standards :

PUSH <s>	équivalent à	**MOVE**	<s>,TST
POP <d>	équivalent à	**MOVE**	TST,<d>
TNIL <s>	équivalent à	**EQ**	<s>,NIL
FNIL <s>	équivalent à	**NEQ**	<s>,NIL

Les MACRO PUSH et POP permettent de manipuler la pile et sont les versions modernes des instructions BURRY et UNBURRY de la 2ème machine de TURING [CARPENTER 76].

3.5.2 Les pseudo-instructions de test

La machine VCMC2 comporte des outils qui permettent de tester son propre fonctionnement. En particulier, il est possible :
- de tracer dynamiquement les instructions exécutées
- de tracer dynamiquement les contenus des registres
- de tracer dynamiquement le contenu de la pile
- d'exécuter un programme VCMC2 de manière incrémentale et d'appeler l'interprète VLISP avant chaque exécution d'instruction.

Ces outils sont décrits au paragraphe suivant. Le positionnement du mode trace et du mode pas-à-pas s'effectue grâce aux pseudo-instructions suivantes qui ne modifient pas le contenu des mémoires ou des registres.

TRACE	positionne le mode trace
UNTRACE	enlève le mode trace
STEP	positionne le mode pas-à-pas
UNSTEP	enlève le mode pas-à-pas
SILENCE	sauve l'état courant des modes et passe en mode non-trace
REVIVE	repasse dans les modes sauvés par le dernier **SILENCE**

3.5.3 Les pseudo-instructions de réservation de mémoire

Il existe deux pseudo-instructions de réservation de mémoire, l'une ne réserve qu'un seul mot, et l'autre un bloc de mots contigus.

```
DATA    objet-VLISP  ou  ( adresse-mémoire )
BLOCK   nombre , objet-VLISP  ou  ( adresse-mémoire )
```

La première réserve un mot mémoire qui est initialisé avec un pointeur sur un objet VLISP ou bien une adresse mémoire du programme. La seconde réserve un bloc de mots contigus dont la taille est spécifiée en position premier opérande, chaque mot doit être initialisé avec le pointeur fourni en position deuxième opérande.

3.5.4 Les pseudo-instructions de déclaration

```
COMMENT
ENTRY    <nom>,<F-TYPE>,<P-TYPE>
```

COMMENT est une déclaration de commentaires et le reste de l'instruction est complétement ignoré.

ENTRY est une déclaration de point d'entrée de fonction standard. <nom> est le nom de la fonction. Ce nom doit être une étiquette du programme. <F-TYPE> est le ***F-TYPE*** de la fonction qui est donné en clair (i.e. 0SUBR 1SUBR 2SUBR 3SUBR NSUBR ou FSUBR). <P-TYPE> est le ***P-TYPE*** de la fonction i.e. un chiffre codé permettant de contrôler l'édition future de cette fonction. Il est décrit au chapitre 6

3.6 UTILISATION DE LA MACHINE VCMC2.

Pour illustrer l'utilisation de la machine VCMC2, nous allons étudier la transcription de la fonction **SUBST** en langage machine VCMC2.

Cette fonction est définie en VLISP de la manière suivante :

```
[1]   (DE SUBST (x y z)
[2]      (IF (ATOM z)
[3]         (IF (EQ z y) x z)
[4]         (CONS (SUBST x y (CAR z))
[5]               (SUBST x y (CDR z)))))
```

C'est une fonction récursive à 3 arguments qui substitue **x** à toutes les occurrences de **y** dans la liste **z**.

```
{1}             ENTRY    SUBST,3SUBR

{2}   SUBST:    TLIST    A3,,[JUMP (SUBST1)]
{3}             EQ       A3,A2,[RETURN]
{4}             MOVE     A3,A1,[RETURN]
{5}   SUBST1:   PUSH     A1
{6}             CDR      A3,TST
{7}             CAR      A3,A3,[CALL (SUBST)]
{8}             POP      A3
{9}             XTOPST   A1,,[CALL (SUBST)]
{10}            CONS     TST,A1,[RETURN]
```

Cette fonction occupe 12 mots VCMC2 (qui possèdent en général 16 bits) *{Note 1}*. Voici la description des instructions :

{1} déclare la fonction **SUBST** comme SUBR à 3 arguments. Cette pseudo-instruction correspond à la déclaration VLISP [1]. A l'entrée de la fonction A1 contient la valeur du 1er argument x, A2 celle de y et A3 celle de z.

{Note 1} ce chiffre est à comparer avec celui obtenu pour traduire la même fonction dans la machine de Peter DEUTSCH [DEUTSCH 73], qui demandait 37 mots de 8 bits.

{2} traduction du test VLISP [2]. La clause *sinon* se trouve en **SUBST1:** .

{3} test [3], la clause *alors* ne nécessite aucun travail car x se trouve encore dans A1 et un [RETURN] peut être utilisé.

{4} clause *sinon* du 2ème test (i.e. z). La valeur retournée par la fonction devant être dans A1, il faut transférer A3 (i.e. z) dans A1 puis réaliser le retour de la fonction.

{5} clause *sinon* du 1er test (i.e. CONS ...). Sauvegarde de A1, qui va être détruit par l'appel récursif de la fonction.

{6} sauvegarde du CDR de A3 (i.e. de z) dans la pile. Cette valeur sera utilisée en 2ème argument du CONS.

{7} 1er appel récursif de la fonction **SUBST**. A1 contient x, A2 contient y et A3 contient (CAR z). A1 et le CDR de A3 ont été sauvés dans la pile. Au retour de cet appel, A2 et A3 seront inchangés et A1 contiendra la valeur de l'appel.

{8} restauration de A3 prêt à être utilisé au 2ème appel récursif.

{9} échange du sommet de pile (i.e. l'ancien A1) avec la valeur du premier appel récursif et 2ène appel récursif de **SUBST**.

{10} calcul de la valeur de **SUBST** par construction d'un doublet dont le CAR est la valeur du 1er appel récursif (sauvé dans la pile) et de CDR la valeur du 2ème appel récursif. Le doublet crée est retourné dans A1 comme valeur de la fonction.

3.7 OUTILS D'ANALYSE ET DE MISE AU POINT.

Le simulateur de la machine VCMC2 comporte des outils d'analyse et de mise au point des programmes écrits en VCMC2.

3.7.1 Les outils d'analyse

Les outils d'analyse permettent d'étudier le comportement du programme simulé à la fois statiquement et dynamiquement.

L'analyse statique porte sur l'étude du code lui-même de l'interprète VLISP écrit en VCMC2 ainsi que des modules d'entrée/sortie (i.e. le code fourni aux appendices **C**, **E** et **F**). Cette analyse permet de réaliser les références croisées (voir les appendices) ainsi que différents comptages tels que :

- le nombre et le type des fonctions définies
- les occurrences des types d'instructions
- les occurrences des types des continuations des instructions
- les occurrences des types des opérandes des instructions

L'analyse dynamique porte sur l'étude des instructions effectivement simulées. Cette analyse fournit le même type de tableaux que l'analyse statique augmentée des informations suivantes :

- nombre de CONS effectués
- taille maximum de la pile
- temps de simulation d'une instruction

Tous les tableaux d'analyse de ce chapitre ont été réalisés par le simulateur en mode statique ou en mode dynamique selon le cas.

Voici les fréquences d'apparition des instructions, en étudiant statiquement le code de l'interprète et des fonctions d'entrées/sorties (i.e. du code des appendices **C**, **E**, et **F**).

Occurrences statiques des instructions

Nombre d'instructions : 1320

1	ADD	= 20	1.52	%	0	%
2	CAR	= 98	7.42	%	0	%
3	CDR	= 124	9.39	%	0	%
4	CONS	= 27	2.05	%	0	%
5	CVAL	= 11	0.83	%	0	%
6	DISPT	= 2	0.15	%	0	%
7	DIV	= 1	0	%	0	%
8	EQ	= 70	5.3	%	0	%
9	FATOM	= 3	0.23	%	0	%
10	FLIST	= 28	2.12	%	0	%
11	FNUMB	= 2	0.15	%	0	%
12	FTYP	= 6	0.45	%	0	%

13	FVAL	=	6	0.45 %	0	%
14	GE	=	7	0.53 %	0	%
15	GT	=	7	0.53 %	0	%
16	IN	=	1	0 %	0	%
17	INTERN	=	3	0.23 %	0	%
18	JUMPX	=	10	0.76 %	0	%
19	LE	=	5	0.38 %	0	%
20	LOGAND	=	1	0 %	0	%
21	LOGOR	=	1	0 %	0	%
22	LOGXOR	=	2	0.15 %	0	%
23	LT	=	6	0.45 %	0	%
24	MOVE	=	475	35.98 %	0	%
25	MOVEX	=	11	0.83 %	0	%
26	MUL	=	1	0 %	0	%
27	NEQ	=	41	3.11 %	0	%
28	NOP	=	85	6.44 %	0	%
29	OUT	=	3	0.23 %	0	%
30	PLEN	=	3	0.23 %	0	%
31	PLIST	=	3	0.23 %	0	%
32	PNAM	=	3	0.23 %	0	%
33	PRINI	=	14	1.06 %	0	%
34	PRSTACK	=	3	0.23 %	0	%
35	PTYP	=	2	0.15 %	0	%
36	READ	=	2	0.15 %	0	%
37	REM	=	1	0 %	0	%
38	SCAR	=	8	0.61 %	0	%
39	SCDR	=	21	1.59 %	0	%
40	SCVAL	=	14	1.06 %	0	%
41	SFTYP	=	5	0.38 %	0	%
42	SFVAL	=	5	0.38 %	0	%
43.	SINDEX	=	12	0.91 %	0	%
44	SPLIST	=	2	0.15 %	0	%
45	SPTYP	=	1	0 %	0	%
46	SSTACK	=	16	1.21 %	0	%
47	STACK	=	14	1.06 %	0	%
48	STOP	=	2	0.15 %	0	%
49	SUB	=	16	1.21 %	0	%
50	SUBFZ	=	1	0 %	0	%
51	TATOM	=	1	0 %	0	%
52	TERPRI	=	1	0 %	0	%
53	TLIST	=	31	2.35 %	0	%
54	TNUMB	=	5	0.38 %	0	%
55	TOPST	=	12	0.91 %	0	%
56	XCONS	=	27	2.05 %	0	%
57	XMOVE	=	5	0.38 %	0	%
58	XTOPST	=	20	1.52 %	0	%

Voici les fréquence d'apparition des instructions dynamiquement observées après exécution du test donné en appendice **G.**

Occurrences dynamiques des instructions

Nb d'instructions exécutées : 80202

1	ADD	= 198	0.25 %	0	%
2	CAR	= 9251	11.53 %	0	%
3	CDR	= 8026	10.01 %	0	%
4	CONS	= 85	0.11 %	0	%
5	CVAL	= 2291	2.86 %	0	%
6	DISPT	= 7420	9.25 %	0	%
7	EQ	= 1983	2.47 %	0	%
8	FATOM	= 124	0.15 %	0	%
9	FLIST	= 1124	1.4 %	0	%
10	FTYP	= 2902	3.62 %	0	%
11	FVAL	= 2903	3.62 %	0	%
12	JUMPX	= 3709	4.62 %	0	%
13	LE	= 1	0 %	0	%
14	LT	= 23	0 %	0	%
15	MOVE	= 18660	23.27 %	0	%
16	MUL	= 5	0 %	0	%
17	NEQ	= 8970	11.18 %	0	%
18	NOP	= 2224	2.77 %	0	%
19	PRINI	= 736	0.92 %	0	%
20	PRSTACK	= 5	0 %	0	%
21	READ	= 103	0.13 %	0	%
22	SCAR	= 8	0 %	0	%
23	SCDR	= 43	0 %	0	%
24	SCVAL	= 1216	1.52 %	0	%
25	SFTYP	= 78	0 %	0	%
26	SFVAL	= 78	0 %	0	%
27	SSTACK	= 670	0.84 %	0	%
28	STACK	= 1534	1.91 %	0	%
29	STOP	= 1	0 %	0	%
30	SUB	= 108	0.13 %	0	%
31	SUBFZ	= 4	0 %	0	%
32	TERPRI	= 498	0.62 %	0	%
33	TLIST	= 1563	1.95 %	0	%
34	TNUMB	= 13	0 %	0	%
35	TOPST	= 555	0.69 %	0	%
36	XCONS	= 202	0.25 %	0	%
37	XTOPST	= 2378	2.97 %	0	%

3.7.2 Les outils de mise au point

Les outils de mise au point de la machine VCMC2 se composent d'une trace et d'une exécution incrémentale (pas-à-pas).

La trace permet d'obtenir avant chaque exécution d'instruction l'impression :
- de l'instruction elle-même
- du contenu des registres et de la pile

Voici un extrait de la trace de l'évaluation de l'expression :

(SUBST 'X 'A '(A B A C))

en utilisant la fonction décrite dans la section précédente. Seuls les registres modifiés sont imprimés par la trace :

```
.....
.....
.....
EVCAR:
     CAR      A1, A1
          A1 = ( QUOTE X )
EVALA1:
     NEQ      NIL, (@ EVALST), [JUMP (EVALT)]
EVALAN:
     MOVE     A1, (@ FORME)
     TLIST    A1, NIL, [JUMP (EVALF)]
EVALF:
     CAR      A1, A2
          A2 = QUOTE
     CDR      A1, A1
          A1 = ( X )
EVALFU:
     FATOM    A2, NIL, [JUMP (EVALS)]
     FVAL     A2, TST
     FTYP     A2, A3
          A3 = 6
EVALIN:
     JUMPX    (TEVAL), A3
EVALFX:
     NOP      NIL, NIL, [RETURN]
QUOTE:
     CAR      A1, A1, [RETURN]
          A1 = X

....
....
```

L'option pas-à-pas permet, avant toute exécution d'instruction, de se retrouver dans un TOPLEVEL de VLISP, ce qui permet de consulter n'importe quel registre ou mémoire, de faire évaluer n'importe quelle expression ou de faire exécuter n'importe quelle instruction VCMC2 *{Note 1}*.

Voici un exemple d'exécution incrémentale de l'interprète durant l'évaluation de l'expression vue précédemment. Avant chaque instruction, la machine VCMC2 imprime le caractère ↑. Si l'utilisateur frappe un espace, l'instruction est exécutée normalement sinon VCMC2 évalue une expression VLISP quelconque. Cette évaluation donne accès à tout l'interprète et permet d'accéder à toutes les variables du simulateur.

```
      .....
      .....
↑     SUBST:
↑
?   A1
    X
↑
?   A2
    A
↑
?   A3
    (A B A C)
↑             FATOM     A3, NIL, [JUMP (SUBST1)]
↑     SUBST1:
↑             MOVE      A1, TST
↑             CDR       A3, TST
↑             CAR       A3, A3, [CALL (SUBST)]
↑     SUBST:
↑
?   A3
    A
↑             FATOM     A3, NIL, [JUMP (SUBST1)]
↑             EQ        A3, A2, [RETURN]
↑             MOVE      TST, A3
↑             XTOPST    A1, NIL, [CALL (SUBST)]
↑
?   A3
    (B A C)
```

{Note 1} la possibilité d'avoir accès à un langage de très haut niveau (en l'occurrence l'interprète VLISP lui-même) donne une puissance inégalée à cette méthode de mise au point dont les possibilités dépassent largement celles du très célèbre DDT de DEC [DEC 75c]. Voir à ce sujet les remarques de M. Model sur la mise au point interactive dans un environnement complexe [MODEL 79].

```
↑     SUBST:
↑           FATOM     A3, NIL, [JUMP (SUBST1)]
↑     SUBST1:
↑           MOVE      A1, TST
↑           CDR       A3, TST
↑           CAR       A3, A3, [CALL (SUBST)]
↑     SUBST:
↑           FATOM     A3, NIL, [JUMP (SUBST1)]
↑           EQ        A3, A2, [RETURN]
↑           MOVE      A3, A1, [RETURN]
↑
?   A1
    X
↑
?   A2
    A
↑
?   A3
    B
↑           MOVE      TST, A3
↑           XTOPST    A1, NIL, [CALL (SUBST)]
↑     SUBST:
↑           FATOM     A3, NIL, [JUMP (SUBST1)]
↑
?   A3
    (A C)
↑     SUBST1:
↑           MOVE      A1, TST
↑           CDR       A3, TST
↑           CAR       A3, A3, [CALL (SUBST)]
↑     SUBST:

      .....
      .....
```

3.8 LES INCARNATIONS DE LA MACHINE VCMC2.

La machine référentielle VCMC2 peut recevoir plusieurs incarnations possibles parmi lesquelles nous verrons :
- la simulation symbolique
- l'interprétation binaire
- la compilation ou la macro-génération.

La simulation symbolique consiste à interpréter directement le code symbolique des programmes écrits en VCMC2. Cet interpréteur n'est réalisable que dans un langage permettant les manipulations symboliques. VLISP s'y prête très évidemment. Ainsi VLISP 10 a été utilisé pour écrire le simulateur symbolique donné à l'appendice B. L'utilisation de VLISP nous a permis d'obtenir un simulateur très court (1000 lignes environ) très rapidement mis en oeuvre (il ne nécessite en effet aucune traduction préalable du code symbolique) et nous y avons inclus des outils de mise au point dynamiques et symboliques. En revanche ce simulateur n'est pas très rapide (de l'ordre de 5 milli-secondes pour simuler une instruction avec le maximum de tests) et demande une grande place en mémoire (6k doublets pour le simulateur de l'appendice B et 8k doublets pour stocker l'interprète sous sa forme symbolique donné à l'appendice F).

Pour diminuer l'espace occupé par les programmes écrits en VCMC2, il est possible d'écrire un assembleur VCMC2 traduisant les programmes écrits en VCMC2 sous la forme la plus compacte, la forme binaire. Toutefois ce nouveau code devra être interprété par un interpréteur binaire spécialisé.

Enfin pour diminuer le temps de simulation, nous avons réalisé un traducteur permettant de macro-générer (voire de compiler) les programmes écrits en VCMC2 dans d'autres machines (PDP10, PDP11, Z80, T1600). Cette macro-génération peut être également réalisée soit à partir de macro-générateurs universels tels le GPM de STRACHEY [STRACHEY 65] ou le STAGE 2 de [POOLE 70, WAITE 73].

CHAPITRE 4

L'EVALUATEUR DE BASE

Le but de ce chapitre est de fournir la description complète de l'implémentation de notre interprète VLISP de base. Nous appelerons évaluateur de base, un évaluateur ne permettant d'évaluer que des formes atomiques (de type symbole atomique ou nombre) ou bien des appels de fonctions (de type SUBR, FSUBR, EXPR, FEXPR ou MACRO).

Il existe en VLISP plusieurs types de fonctions.

La pluralité des types est déterminée par :

1) le langage d'écriture de la fonction.
   Nous distinguerons deux langages d'écriture :
   - le langage VCMC2. Ces fonctions, appelées les SUBR, sont résidentes dans le système dès sa mise en route. L'utilisation du langage machine leur confère une vitesse d'exécution maximum. Ces fonctions, qui sont également appelées <u>fonctions standards</u>, sont écrites en langage machine parcequ'il n'est pas possible de les écrire en VLISP (comme par exemple les primitives CAR ou CONS) ou bien parcequ'elles sont très fréquemment utilisées.
   - VLISP pour la plupart des fonctions définies par l'utilisateur.

2) le type d'évaluation des arguments :
   les arguments peuvent être évalués ou non à l'entrée de la fonction. En règle générale, tous les arguments d'une fonction sont évalués. Toutefois pour certaines fonctions aucun argument n'est évalué ce qui permet de construire des fonctions de contrôle ou des fonctions conditionnelles. Par exemple pour la fonction conditionnelle IF, les arguments ne sont pas évalués à l'entrée de la fonction mais ils le sont sélectivement à l'intérieur de cette fonction.

3) le nombre d'arguments.
Le nombre des arguments des SUBR est connu à l'avance : ce nombre peut être :
- fixe
  dans le cas des SUBR qui évaluent leurs arguments on distingue les fonctions :
  - sans argument
  - à 1 argument
  - à 2 arguments
  - à 3 arguments

  Nous appelerons respectivement ces fonctions 0SUBR, 1SUBR, 2SUBR et 3SUBR
- variable. ces SUBR s'appellent des NSUBR.

dans le cas des SUBR qui n'évaluent pas leurs arguments, ils sont en nombre indéfini et ces fonctions se nomment des FSUBR.

Les fonctions écrites en VLISP ont un nombre d'arguments variable. Si les arguments sont évalués, ces fonctions sont appelées des EXPR et s'ils ne le sont pas, elles sont appelées des FEXPR ou des MACRO en fonction du type de traitement désiré : les MACRO étant évaluées deux fois pour permettre un traitement préalable de l'appel de la fonction.

Notre modèle de base connaît donc 9 types de fonctions, qui sont :

```
les 0SUBR : SUBR à 0 argument
les 1SUBR : SUBR à 1 argument évalué
les 2SUBR : SUBR à 2 arguments évalués
les 3SUBR : SUBR à 3 arguments évalués
les NSUBR : SUBR à N arguments évalués
les FSUBR : SUBR à N arguments non-évalués
les  EXPR : qui possèdent N arguments évalués
les FEXPR : qui possèdent N arguments non-évalués
les MACRO : qui possèdent N arguments non-évalués
```

4.1 COMMENT CARACTERISER UNE FONCTION ? LE COUPLE *F-TYPE F-VAL.*

En VLISP, une fonction est caractérisée par un couple composé du type de la fonction tel qu'il vient d'être vu et de la valeur de définition de la fonction.

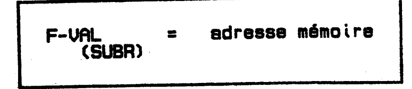
Le type d'une fonction est représenté par un nombre *{Note 1}* que nous appelerons le *F-TYPE* d'une fonction. Voici les codes des *F-TYPES* de notre modèle de base :

Code des F-types

type de la fonction	F-type
OSUBR	1
1SUBR	2
2SUBR	3
3SUBR	4
NSUBR	5
FSUBR	6
EXPR	7
FEXPR	8
MACRO	9
ce n'est pas une fonction	0

Un *F-TYPE* égal à 0 signifie que l'on n'a pas affaire à une fonction.

En plus d'un type chaque fonction possède une valeur de définition : la *F-VAL*. Dans le cas des SUBR, cette *F-VAL* est l'adresse en mémoire du sous-programme écrit en VCMC2 réalisant la fonction.


F-VAL (SUBR) = adresse mémoire

{Note 1} notre modèle utilise un nombre pour pouvoir effectuer rapidement des aiguillages sur ce type au moyen de branchements indirects indexés.
Cet aiguillage est réalisé par l'instruction VCMC2 : JUMPX.

Dans le cas des EXPR, des FEXPR et des MACRO, cette ***F-VAL*** est une liste de la forme :

```
F-VAL              =    ( <svar> <e1> ... <eN> )
   (EXPR/FEXPR/MACRO)
```

Le premier argument de cette liste **<svar>** est la spécification des variables de la fonction.
Les éléments suivants de cette liste **<e1>** ... **<eN>** sont les différentes expressions du corps de la fonction, qui seront évaluées en séquence lors de l'évaluation de la fonction.

4.2 COMMENT DECRIRE ET INVOQUER LES FONCTIONS ?

Le couple ***F-TYPE, F-VAL*** peut être utilisé directement (au moyen d'une forme INTERNAL) pour décrire une fonction, il s'agit dans ce cas de fonction anonyme ou bien il peut être associé à un atome, nous parlerons dans ce cas de fonction nommée.

La nouvelle forme INTERNAL va nous permettre de décrire une fonction anonymement sans l'associer à un atome particulier ***{Note 1}***.

(INTERNAL ***F-TYPE F-VAL***)

est la représentation d'une fonction anonyme dont on spécifie à la fois son ***F-TYPE*** et sa ***F-VAL***.

Par exemple :

(INTERNAL 2 10000)

est la représentation d'une SUBR à 2 arguments évalués. Le sous-programme traitant la fonction se trouve à l'adresse 10000.

(INTERNAL 7 ((I J) (PRINT I J) (+ I J)))

est la représentation d'une EXPR à 2 arguments évalués. La liste des variables de la fonction est (I J) et le corps de la fonction est ((PRINT I J) (+ I J))

Il existe, dans le cas des EXPR, une autre forme de description, la forme LAMBDA, que nous avons gardé dans un souci d'historicité et de compatibilité avec les autres interprètes VLISP.

{Note 1} le modèle VLISP est le seul interprète LISP n'obligeant pas à nommer les fonctions de type SUBR ou FEXPR.

la forme :

(LAMBDA <svar> <e1> ... <eN>)

correspond strictement à :

(INTERNAL 7 (<svar> <e1> ... <eN>))

Le couple ***F-TYPE F-VAL***, peut être associé à un atome. Cette association est réalisée à l'initialisation de l'interprète dans le cas des fonctions standards, ou bien au moyen des fonctions de définition pour les fonctions de l'utilisateur.

Un appel de fonction se réalise par l'évaluation d'une liste de la forme :

(<fnt> <a1> ... <aN>>)

dans laquelle **<fnt>** est soit une fonction anonyme de type INTERNAL ou LAMBDA, soit un symbole atomique auquel est associé une fonction, **<a1>** ... **<aN>** est la liste des arguments de l'appel de la fonction.

Enfin VLISP possède une dernière forme pour invoquer les fonctions, la forme **LET**, si la fonction est une fonction anonyme de type LAMBDA dont la spécification des variables **<svar>** est une liste de variables **<v1> ... <vN>**.
Cette forme, qui est une MACRO standard VLISP *{Note 1}*, permet de simplifier l'écriture et la lecture d'un appel de fonction de type **LAMBDA** en accolant les couples variable-valeur.

l'invocation :

```
(LET ((<v1> <a1>) ... (<vN> <aN>)) <e1> ... <eN>)
```

est automatiquement transformée en :

```
((LAMBDA (<v1> ... <vN>) <e1> ... <eN>) <a1> ... <aN>)
```

{Note 1} Voici la définition en VLISP de cette MACRO :

```
(DM LET (l)
   (RPLACB l
     (COND
       ((NULL (CADR l))
          (CONS (CONS LAMBDA (CONS () (CDDR l)))))
       ((ATOM (CAADR l))
          (CONS (MCONS LAMBDA (CONS (CAADR l)) (CDDR l))
             (CDADR l)))
       (T (CONS (MCONS LAMBDA (MAPCAR (CADR l) 'CAR) (CDDR l))
             (MAPCAR (CADR l) 'CADR)))))))
```

4.3 LA LIAISON DES ARGUMENTS.

A chacun des types de fonctions vues précédemment, est associé un type spécifique de liaison des arguments. Cette liaison va assigner dans la machine un emplacement à ces arguments.

Dans le cas des SUBR, les arguments sont liés à des registres de la machine VCMC2 et dans le cas des EXPR des FEXPR ou des MACRO, les arguments sont liés aux variables de la fonction.

4.3.1 La liaison des SUBR

La présence des registres dans la machine VCMC2 nous permet d'utiliser au mieux les possibilités matérielles de la machine en minimisant le temps d'accès aux arguments et l'occupation mémoire des instructions manipulant ces arguments.

Le nombre d'arguments d'une fonction n'étant pas limité, les quatre registres de la machine se révèlent insuffisants pour stocker tous les arguments. Toutefois le nombre des arguments des SUBR standards est en général peu élevé (0, 1, 2 ou 3 arguments). Voici les occurrences des SUBR standards de notre modèle :

Occurrences statiques des SUBR

Nombre de fonctions SUBR : 161

1	0SUBR	= 10	6.21 %	0	%
2	1SUBR	= 54	33.54 %	0	%
3	2SUBR	= 46	28.57 %	0	%
4	3SUBR	= 5	3.11 %	0	%
5	NSUBR	= 4	2.48 %	0	%
6	FSUBR	= 42	26.09 %	0	%

Ce tableau nous montre clairement quels types de SUBR doivent être priviligiés : les 1SUBR, 2SUBR et FSUBR qui représentent près de 90% des fonctions.

Il va donc être possible d'utiliser au mieux les quatre registres de la machine, dans le cas des SUBR, de la manière suivante :

F-type	Contenus des registres
1 (0SUBR)	Il n'y a pas d'argument, aucun registre ne contient de valeur utilisable.
2 (1SUBR)	A1 ← la valeur du 1er argument
3 (2SUBR)	A1 ← la valeur du 1er argument A2 ← la valeur du 2ème argument
4 (3SUBR)	A1 ← la valeur du 1er argument A2 ← la valeur du 2ème argument A3 ← la valeur du 3ème argument
5 (NSUBR)	A1 ← la liste des valeurs de tous les arguments
6 (FSUBR)	A1 ← la liste des arguments non-évalués

Toutes les fonctions de type SUBR retournent leur valeur également dans le registre A1, ce qui nous permet de ne faire aucune manipulation de registre entre les appels imbriqués de fonctions de type 0SUBR, 1SUBR, FSUBR ou NSUBR (près de 70% des SUBR) car la valeur de la fonction calculée qui se trouve dans A1 devient l'argument de la fonction suivante qui une fois calculée devient l'argument de la fonction suivante ...etc... Cette allocation de registre est optimale (du point de vue manipulation de registres) et permet d'utiliser le registre A1 comme <u>registre pipe-line</u> pour les SUBR :

La composition :

```
(FREVERSE (EVLIS (READ)))
```

est équivalente à cette suite d'appels de sous-programmes :

```
NOP     ,,[CALL (READ)]
NOP     ,,[CALL (EVLIS)]
NOP     ,,[CALL (FREVERSE)]
```

qui ne nécessite aucune manipulation de registre entre les sous-programmes.

Ce type de liaison se révèle donc très efficient mais reste limité aux fonctions standards SUBR qui doivent être écrites en langage machine.

4.3.2 La liaison des EXPR/FEXPR/MACRO

Dans les fonctions de type EXPR, FEXPR et MACRO, les arguments vont être liés aux variables de la fonction. Ces variables sont décrites dans la spécification de variables <svar> qui est le premier élément de la description de la fonction. Les arguments <a1> ...<aN> sont rassemblés dans une liste qui est le CDR de l'appel de la fonction. Ces arguments sont évalués pour les fonctions de type EXPR mais pas pour les FEXPR ni les MACRO.

```
fonction    :  ( <svar>  <e1>  ...  <eN> )
invocation  :  ( <fonc>  <a1>  ...  <aN> )
```

Dans le cas des fonctions de type EXPR, quatre cas peuvent se présenter :

1) la spécification de variables <svar> est nulle. En ce cas aucun argument n'est évalué, aucune liaison n'est réalisée et le corps de la fonction peut être exécuté directement.

```
fonction    :  (  ()      <e1>   ...   <eN> )
invocation  :  ( <fonc> )
liaison     :  aucune
exécution   :  (PROGN <e1> ... <eN> )
```

2) la spécification de variables <svar> est une liste de variables. En ce cas, les variables de cette liste sont liées une à une aux différents arguments évalués avant la réalisation des liaisons.

```
fonction    :  ( ( <v1> ... <vN> ) <e1>  ...  <eN> )
invocation  :  ( <fonc>  <a1> ... <aN> )
liaisons    :  <v1> ← (EVAL <a1>) ... <vN> ← (EVAL <aN>)
exécution   :  (PROGN <e1>  ...  <eN>)
```

Toutes les évaluations des arguments doivent s'opérer AVANT les liaisons proprement dites.

En effet :

```
(LET (x 2)
   (LET ((x (PLUS x 1)) (y (PLUS x x)))
      (PLUS x y)))
```

☞ 7

la liaison séquentielle de x puis de y aurait livrée

☞ 9

Enfin, si le nombre d'arguments fourni à l'appel est plus grand que le nombre de variables de **<svar>**, ils sont ignorés SANS ETRE EVALUES. Si le nombre d'arguments est inférieur au nombre des variables, les variables restantes sont liées à la valeur NIL.

```
((λ (x y z) ...) 10)
```

va réaliser les liaisons :

```
x   ←   10
y   ←   NIL
z   ←   NIL
```

3) la spécification de variables <svar> contient une variable simple. En ce cas c'est la liste de toutes les valeurs des arguments qui est liée à cette unique variable.

```
fonction     :  ( <v>    <e1>   ...   <eN> )
invocation   :  ( <fonc>  <a1> ...  <eN> )
liaison      :  <v> ← [ <a1> ... <aN> ]   {Note 1}
exécution    :  (PROGN  <e1>   ...   <eN>)
```

4) il existe en outre en VLISP un quatrième type de spécification de variables qui regroupe les 2 cas vus précédemment, et qui est représenté par une liste pointée de variables. En ce cas les premières variables sont liées une à une avec les valeurs des arguments correspondants, et la dernière variable, qui se trouve en

{Note 1} la notation VLISP [<e1> ... <eN>] *correspond à la description d'une liste évaluée (voir le paragraphe 5.2).*
[<e1> ... <eN>] *est équivalent à* (LIST <e1> ... <eN>)

position dernier CDR de la spécification des variables, est liée avec une liste regroupant toutes les valeurs du reste des arguments.

```
fonction    :  ( (<v1> ... <vN-1> . <vN> ) <e1>   ...   <eN> )
invocation  :  ( <fonc>  <a1> ... <aN-1> <aN> <aN+1> ... <aM> )
liaisons    :  <v1> ←  (EVAL <a1>) ... <vN-1> ← (EVAL <aN-1>)
               et   <vN> ← [ <aN+1> ... <aM> ]
exécution   :  (PROGN  <e1>  ...  <eN>)
```

Dans le cas des FEXPR, la spécification des variables <svar> doit être une liste d'au moins une variable et c'est la liste des arguments elle-même, non évaluée, qui est liée à la première variable de la liste <svar>. Si cette liste possède d'autres variables, elles sont liées à la valeur NIL et font ainsi office de variables locales.

```
fonction    :  ( (<v1> ... <vN> ) <e1>  ...   <eN> )
invocation  :  ( <fonc>  <a1> ... <aN> )
liaisons    :  <v1> ← ( <a1>  ... <aN> )  et
               <v2> ← NIL    ...   <vN> ← NIL
exécution   :  (PROGN  <e1>  ...  <eN>)
```

Ce type de fonction est souvent utilisé pour créer des fonctions de contrôle qui n'évaluent pas tous leurs arguments mais le font sélectivement.

Dans le cas des fonctions de type MACRO, seule la première variable est liée comme dans le cas des FEXPR. Toutefois c'est l'appel de la MACRO tout entier qui est lié à la première variable. Les autres variables de la liste étant liées à la valeur NIL et faisant office également de variables locales.

```
fonction    :  ( (<v1> ... <vN> ) <e1>  ...  <eN> )
invocation  :  ( <fonc>  <a1> ... <aN> )
liaisons    :  <v1> ← ( <fonc> <a1>  ... <aN> )  et
               <v2> ← NIL   ...   <vN> ← NIL
exécution   :  (EVAL (PROGN <e1> ... <eN>))
```

Outre la liaison, les fonctions de type MACRO se distinguent des FEXPR et des EXPR par la manière dont est évalué le corps. La valeur d'une fonction de type MACRO est la valeur de la valeur de l'évaluation du corps. Il se produit donc un mécanisme de double évaluation ce qui permet dans un premier temps d'expanser la MACRO et dans un second temps de calculer sa valeur.

4.4 TABLEAU SYNOPTIQUE DE L'EVALUATEUR DE BASE.

Nous allons à présent décrire le fonctionnement interne de l'évaluateur de base, la fonction **EVAL**. Voici le tableau synoptique des différents modules de **EVAL** :

Objet à évaluer :	Fréquence et nom du module : *{Note 1}*
Symbole atomique : une variable	34 % EVALAT
Nombre : une constante numérique	6 % EVALNB
Liste : retourne la valeur de l'application de la fonction aux arguments	60 % EVALIS
cette fonction peut-être :	
un symbole atomique :	59 % EVALFA
type de la fonction :	
0SUBR	<1 % EVAL0
1SUBR	23 % EVAL1
2SUBR	11 % EVAL2
3SUBR	<1 % EVAL3
NSUBR	<1 % EVALN
FSUBR	15 % EVALF
EXPR	10 % EVEXP
FEXPR	<1 % EVFEXP
MACRO	<1 % EVMAC
un nombre	<1 % EVALFN
une liste qui est :	<1 % EVALFS
une forme INTERNAL	<1 % EVALL
une forme LAMBDA	<1 % EVALI
une liste à évaluer	<1 %

{Note 1} ces fréquences ont été obtenues après analyse du test donné en appendice G.

4.5 ACCES AUX VARIABLES ET EVALUATION DES ATOMES.

En VLISP comme dans tous les LISP les variables des fonctions sont fluides : les liaisons et les portées des variables sont dynamiques. A l'entrée d'une fonction les paramètres actuels (les valeurs des arguments) sont liés aux paramètres formels (les variables), et au sortir de la fonction les variables doivent retrouver leurs anciennes valeurs.

Plusieurs stratégies sont disponibles pour réaliser ce type de liaisons. Les principaux problèmes à résoudre sont :

- la facilité de la fabrication de la liaison
- la facilité de la destruction de la liaison
- la facilité d'accès aux variables liées
- l'occupation mémoire résultant de cette liaison

4.5.1 Les liaisons par A-listes

Les premières implantations (LISP 1.5 [McCARTHY 62]) utilisaient une A-liste ou liste d'associations pour réaliser les liaisons. Cette A-liste contenait l'ensemble de toutes les liaisons des variables sous la forme d'une liste de type :

```
A-liste = ((variable . valeur) ... (variable . valeur))
```

Lier une variable consistait à ajouter en tête de la A-liste une nouvelle association représentée par un doublet de liste comprenant le nom de la variable en partie CAR et sa nouvelle valeur en partie CDR.

Délier une variable consistait à enlever de la A-liste le doublet de liste correspondant.

L'accès à une variable liée se faisait par consultation de la A-liste jusqu'à rencontre d'un doublet contenant en partie CAR le nom de la variable. La valeur de la variable était trouvée en partie CDR de ce même doublet.

Les performances de ce modèle étaient bonnes du point de vue création et restitution des liaisons, étaient moins bonnes du point de vue occupation mémoire qui nécessitait 2 doublets de liste par liaison (ces doublets étaient alloués dans l'espace liste ce qui augmentait le nombre d'appels du récupérateur et ralentissait d'autant plus le système) mais surtout les performances étaient catastrophiques du point de vue de l'accès aux valeurs des variables : en cas de récursion profonde, la A-liste devenait très longue et les temps d'accès devenaient prohibitifs (l'accès à une variable globale au fond d'une récursion profonde était proportionnel à la profondeur de la

récursion!) *{Note 1}*.

En conséquence ces types d'interprètes distinguaient deux types de variables :

- les variables de la A-liste affectables par liaison d'arguments de fonctions et par les fonctions d'affectation de type SET et SETQ

- les variables globales (appelées SPECIAL) accessibles par leurs *C-VAL* par les fonctions de type CSET et CSETQ.

Cette distinction permettait d'accélérer l'accès aux valeurs des variables globales qui s'effectuaient par consultation de la *C-VAL*.

4.5.2 La liaison superficielle

Dès 1966 [WHITE 78] le modèle utilisant une A-liste fut abandonné au profit d'un mécanisme de sauvetage et de restauration utilisant une vraie pile. Dans ce modèle, appelé liaison superficielle (shallow binding), l'accès aux variables s'opère toujours par consultation de la *C-VAL* de la variable. Toutes les variables sont globales au sens de LISP 1.5. La liaison superficielle se réalise de la facon suivante :

1 - évaluation (ou non) des valeurs des arguments
2 - sauvegarde dans la pile des anciennes *C-VAL* des variables de la fonction
3 - modification des *C-VAL* des variables avec les arguments

Puis au retour de la fonction, les variables reprennent leurs anciennes valeurs sauvées dans la pile.

Cette méthode est moins performante que la méthode précédente utilisant une A-liste, lorsqu'il s'agit de réaliser la liaison proprement dite, car il faut sauver dans la pile les anciennes *C-VAL* des variables de la fonction. De même la restauration en fin de la fonction nécessite un temps non négligeable. Toutefois le très grand intérêt de cette méthode réside dans l'accès direct aux valeurs des variables qui est direct. La simple consultation de la *C-VAL* d'une variable suffit. Cet accès prend un temps fixé d'avance quelquesoit la profondeur de la récursion. L'autre avantage de cette méthode est de n'occuper que 2 mots (1 mot pour le nom de la variable et 1 mot pour son ancienne valeur) par liaison de variable dans l'espace pile, lequel n'a pas besoin d'être récupéré avec des procédés couteux comme le récupérateur de doublets de liste.

Le gros défaut de cette méthode par rapport à la A-liste est de ne pas pouvoir traiter les objets FUNARG complets tels qu'ils sont décrits dans [WEIZENBAUM 68, MOSES 70].

{Note 1} les essais d'accélération de l'accès avec ce type de liaison [BAKER 77a] ne résolvaient pas le problème d'occupation mémoire.

4.5.3 L'évaluation des atomes

Notre modèle utilise la liaison superficielle pour lier les variables des fonctions.
Voici le début de la fonction interprète EVAL qui traite des atomes.

Evaluation des atomes

```
{1}   EVAL:
{2}          MOVE     A1,(@ FORME)
{3}          DISPT    (TEVAL1),A1
      TEVAL1:
{4}          DATA     (EVALAT)
{5}          DATA     (EVALNB)
{6}          DATA     (EVALIS)
      EVALAT:
{7}          CVAL     A1,A1
{8}          NEQ      A1,'UNDEF,[RETURN]
             erreur "variable indéfinie" le
             nom de l'atome se trouve dans
             le mot FORME.
      EVALNB:
{9}          NOP       ,,[RETURN]
```

{1} EVAL: est l'étiquette du sous-programme qui évalue une forme quelconque se trouvant dans A1. Ce sous-programme retourne la valeur de l'expression également dans le registre A1.

{2} le premier travail d'EVAL est de sauver dans un mot mémoire de nom FORME la forme qui doit être évaluée. Cette sauvegarde va permettre de détruire le contenu de A1 sans perdre la trace de la forme à évaluer.

{3} l'instruction DISPT réalise le 1er aiguillage sur type d'objet de EVAL et permet de se brancher à l'un des 3 modules de EVAL : 1) EVALAT qui évalue les symboles atomiques, 2) EVALNB qui évalue les nombres, 3) EVALIS qui évalue des listes.

{4,5,6} contient la table d'aiguillage sur type de l'instruction précédente.

{7} arrivé à cette instruction, A1 contient donc une variable (un symbole atomique). Sa valeur est le contenu de sa ***C-VAL***, qui est rangée dans A1, prête à être retournée comme valeur de EVAL. L'accès à la valeur ne nécessite qu'une instruction. L'utilisation d'une A-liste aurait nécessité le sous-programme suivant : (le point d'entrée est **ASSO:**)

```
        A1 ← l'atome, A2 ← la A-liste

ASSO1:  CAR    A2,A3     ; A3 ← couple suivant
        CAR    A3,A4     ; A4 ← variable
        EQ     A4,A1,[JUMP (ASSO2)] ; trouvé
        CDR    A2,A2     ; au suivant
ASSO:   TLIST A2,,[JUMP (ASSO1)]
        {l'atome n'existe pas}
ASSO2:  CDR    A3,A1      ; A1 ← la valeur
```

{8} ce retour ne pourra toutefois se faire que si la ***C-VAL*** de l'atome considéré possède une valeur définie. Tous les atomes possèdent à leur création une valeur indéfinie représentée par le symbole spécial UNDEF. Ce test permet de détecter toute tentative d'accès à une variable, qui n'est ni initialisée ni liée. Le nom de la variable ne se trouve plus que dans le mot de sauvegarde FORME.

{9} cette instruction traite des nombres. Les nombres en LISP n'étant pas évalués, EVAL retourne directement la valeur du nombre.

Ce début de la fonction EVAL montre bien la remarquable efficacité de la liaison superficielle : 4 instructions VCMC2 suffisent pour évaluer un atome.
Ces 4 instructions pourraient être aisément microprogrammées sur une autre machine tel le PDP11/40 [GRIGNETI 76] pour réaliser ainsi une nouvelle instruction machine, EVAL, qui évaluerait son opérande et provoquerait une interruption logicielle en cas d'accès à une variable indéfinie et en cas d'évaluation d'une fonction. Toutefois, la microprogrammation complète de l'interprète EVAL peut difficilement se réaliser actuellement pour des raisons de place dans le micro-code.

4.6 EVALUATION DES APPELS DE FONCTIONS.

Lorsque la forme à évaluer est une liste, la fonction interprète EVAL suppose qu'il s'agit d'un appel de fonction. Le CAR de la forme contient la fonction et le CDR les arguments de cette fonction.

Il existe une grande variété de fonctions, EVAL va devoir réaliser un grand nombre de tests. Toutefois l'utilisation conjugée du ***F-TYPE*** des symboles atomiques et des instructions machine VCMC2 **JUMPX** et **DISPT** va considérablement accélérer les tests en autorisant l'utilisation d'aiguillages.

Les fonctions (i.e. les CAR des formes) peuvent être des symboles atomiques, des nombres ou des listes.

4.6.1 Evaluation des fonctions atomiques

Du fait de leur très grande fréquence (plus de 90% des fonctions), le traitement de ces fonctions doit être extrêmement rapide et ne pas consommer de ressources de type pile, ni de doublets de liste dont la récupération est très laborieuse.
Voici la partie de EVAL (commune au traitement de toutes les fonctions) qui va réaliser un aiguillage contrôlé par le type de la fonction.

```
EVALIS:
        CAR     A1,A2
        CDR     A1,A1
EVALFU:
        DISPT   (TEVAL2),A2
TEVAL2:
        DATA    (EVALFA)
        DATA    (EVALFN)
        DATA    (EVALFS)
EVALFA:
        FVAL    A2,TST
        FTYP    A2,A3
EVALIN:
        JUMPX   (TEVAL3),A3
TEVAL3:
        DATA    erreur     ; 0 : erreur ;
        DATA    (EVAL0)    ; 1 : SUBR à 0 argument ;
        DATA    (EVAL1)    ; 2 : SUBR à 1 argument ;
        DATA    (EVAL2)    ; 3 : SUBR à 2 arguments ;
        DATA    (EVAL3)    ; 4 : SUBR à 3 arguments ;
        DATA    (EVALN)    ; 5 : SUBR à N arguments ;
        DATA    (EVALF)    ; 6 : FSUBR ;
        DATA    (EVEXP)    ; 7 : EXPR ;
        DATA    (EVFEXP)   ; 8 : FEXPR ;
        DATA    (EVMAC)    ; 9 : MACRO ;
```

{10} EVALIS: est le nom de la partie d'EVAL qui traite des fonctions.

{11} La première chose à faire est de séparer la fonction des arguments de l'appel. A2 contient la fonction.

{12} A1 contient la liste des arguments de l'appel. L'appel intégral a disparu, reste qu'il a été précédemment sauvé dans le mot mémoire FORME (voir *{2}*).

{13} EVALFU: est le point d'entrée de EVAL pour lequel A2 contient la fonction et A1 contient la liste des arguments.

{14} ce premier aiguillage sur le type de la fonction permet d'exécuter un des trois modules suivants : 1) EVALFA qui traite des fonctions associées à un symbole atomique, 2) EVALFN qui traite des fonctions numériques, 3) EVALFS qui traite des fonctions spéciales.

{15} table d'étiquettes de l'aiguillage précédent.

{17} la fonction étant associée à un symbole atomique, préparons le lancement de l'évaluation de sa valeur de définition (sa *F-VAL*). Celle-ci est empilée et va permettre le lancement des fonctions de type SUBR en exécutant une continuation **RETURN1**.

{18} puis le ***F-TYPE*** du symbole atomique est chargé dans le registre A3. Ce ***F-TYPE*** contient le type codé de la fonction associée au symbole.

{19} ce point d'entrée sera utilisé pour traiter les fonctions de type INTERNAL.

{20} enfin on réalise un branchement indirect (dans la table d'étiquettes TEVAL3) indexé par le ***F-TYPE*** (contenu dans A3). Ce branchement permet de se retrouver directement dans la routine qui traite un type de fonction spécifique. A l'entrée de chacunes de ces routines, A1 contient toujours la liste des arguments, A2 le nom du symbole de la fonction, et la ***F-VAL*** de la fonction est empilée. Ce type d'aiguillage est trés rapide et n'est pas dépendant du nombre de ***F-TYPE***. Il faut noter que 8 instructions seulement ont été exécutées depuis l'entrée dans la fonction EVAL.

{21}... TEVAL3 est la table des adresses des routines spécialisées de traitement de chacun des types de fonctions.

4.6.2 Le traitement des SUBR

Chaque type de SUBR possède un module spécial qui permet de lançer la fonction spécifiée. A l'entrée de ces modules de lancement, A1 contient la liste des arguments non-évalués (i.e. le CDR de l'appel cf: *{12}*) et la ***F-VAL*** de la fonction (i.e. l'adresse mémoire de la fonction) est empilée (cf: *{17}*).

1) Pour les OSUBR, il n'y a pas d'argument à évaluer et le branchement peut s'effectuer tout de suite. L'adresse de branchement a été empilée en *{17}*, il suffit donc de réaliser une continuation **[RETURN]**.
   De même pour les FSUBR, la liste des arguments non-évaluée se trouvant déjà dans A1, il suffit de réaliser une continuation **[RETURN]** pour lancer la fonction.

```
        Lancement d'une OSUBR
                 ou
        Lancement d'une FSUBR

             EVALO:
             EVALF:
{31}                   NOP,,[RETURN]
```

2) Pour les 1SUBR, il faut au préalable évaluer le 1er argument avant de réaliser un branchement vers la fonction. L'adresse de lancement de la fonction se trouvant empilée, il est possible d'utiliser le JRST-hack *{Note 1}* pour évaluer le 1er argument au moyen d'un JUMP à la fonction EVAL qui déposera dans A1 la valeur de l'argument. Le retour d'EVAL s'effectue directement dans la fonction 1SUBR qui doit être lançée.

{Note 1} le JRST-hack est la transformation itérative d'une récursion terminale dans un langage machine (le nom JRST provient du mnémonique du branchement inconditionnel sur PDP10). En VCMC2 cette transformation s'opère comme suit :

```
---   ,,[CALL (x)]    se transforme en    ---   ,,[JUMP (x)]
NOP   ,,[RETURN]
```

sachant que x s'achève lui-même par NOP ,,[RETURN].

```
              Lancement d'une 1SUBR

{32}  EVAL1:
{33}        CAR A1,A1,[JUMP (EVAL)])
```

3) Pour les 2SUBR, il faut évaluer 2 arguments avant de lancer la fonction et placer ceux-ci respectivement dans les registres A1 et A2. Ceci est réalisé par le code suivant :

```
              Lancement d'une 2SUBR

{34}  EVAL2:
{35}     CDR    A1,TST,[CALL (EVAL1)]
{36}     XTOPST A1,,[CALL (EVAL1)]
{37}     MOVE   A1,A2
{38}     POP    A1,,[RETURN]
```

{35} évalue le 1er argument après avoir sauvé dans la pile le reste des arguments

{36} échange dans le sommet de la pile la valeur du 1er argument avec le reste des arguments puis évalue le 2ème argument

{37} transfère dans A2 la valeur du 2ème argument

{38} récupère la valeur du 1er argument dans A1 et se lance à l'adresse de la fonction de type 2SUBR (i.e. la ***F-VAL*** empilée en *{17}*)

4) Pour les 3SUBR, il faut évaluer 3 arguments avant de lancer la fonction et placer ceux-ci respectivement dans les registres A1, A2 et A3. Ceci est réalisé d'une manière analogue au lancement des 2SUBR :

Lancement d'une 3SUBR

```
{39} EVAL3:
{40}     CDR    A1,TST,[CALL (EVAL1)]
{41}     XTOPST A1
{42}     CDR    A1,TST,[CALL (EVAL1)]
{43}     XTOPST A1,,[CALL (EVAL1)]
{44}     MOVE   A1,A3
{45}     POP    A2
{46}     POP    A1,,[RETURN]
```

{40} évalue le 1er argument après avoir mis à l'abri dans la pile le reste des arguments

{41} échange dans le sommet de la pile la valeur du 1er argument avec le reste des arguments puis évalue le 2ème argument

{42} évalue le 2ème argument après avoir sauvé dans la pile le reste des arguments

{43} échange dans le sommet de la pile la valeur du 2ème argument avec le reste des arguments puis évalue le 3ème argument

{44} transfère dans A3 la valeur du 3ème argument

{45} récupère la valeur du 2ème argument.

{46} récupère la valeur du 1er argument dans A1 et se lance à l'adresse de la fonction de type 3SUBR (i.e. la ***F-VAL*** empilée en *{17}*)

5) Pour les NSUBR, il faut fabriquer dans A1 la liste des valeurs des arguments avant de lancer la fonction. Ceci est réalisé par le sous-programme récursif suivant :

Lancement d'une NSUBR

```
{50} EVALN:
{51}     TNIL    A1,,[RETURN]
{52}     CDR     A1,TST,[CALL (EVAL1)]
{53}     XTOPST  A1,,[CALL (EVALN)]
{54}     CONS    TST,A1,[RETURN]
```

Ce sous-programme est identique à celui traitant les fonctions EVLIS ou LIST.

{51} s'il n'y a pas (ou plus) d'arguments à évaluer lancement de la NSUBR (ou retour du sous-programme récursif en *{54}*).

{52} évaluation de l'argument suivant après avoir sauvé dans la pile le reste des arguments.

{53} échange de la valeur de l'argument avec le reste des arguments et appel récursif de EVALN pour évaluer le reste des arguments

{54} fabrication de la liste et retour à un niveau inférieur d'appel de EVALN ou bien lancement de la NSUBR.

4.6.3 L'évaluation des fonctions de type EXPR

L'évaluation d'une fonction de type EXPR doit s'opérer en trois temps :

1) liaison des variables de la fonction aux arguments fournis à l'appel de cette fonction et sauvetage des anciennes valeurs des variables

2) exécution du corps de la fonction

3) restauration des anciennes valeurs des variables sauvées en 1).

La valeur d'une variable est contenue à tout moment dans sa *C-VAL*. Le sauvetage des anciennes *C-VAL* des variables est réalisé par la construction d'un bloc de sauvegarde dans la pile.

Voici l'état de la pile AVANT d'effectuer la sauvegarde :

```
SP →  | adresse de retour de EVAL |
      .          .....            .
```

Le bloc de sauvegarde aura la forme suivante :

```
SP →  | le nom de la variable N    |
      | son ancienne C-VAL         |
      .          ......            .
      | le nom de la variable 1    |
      | son ancienne C-VAL         |
      |          ¤MARK¤            |
      | adresse de retour de EVAL  |
      .          ......            .
```

Le marqueur ¤MARK¤ sert à indiquer la fin des couples variable-valeur et est utilisé durant la phase de restauration des anciennes valeurs.

Il existe deux méthodes pour construire ce bloc. La première méthode consiste à évaluer tout d'abord tous les arguments et à rassembler toutes les valeurs dans une liste (travail effectué par le sous-programme EVALN:) puis à réaliser les liaisons en parcourant simultanément la liste des variables et la liste des valeurs.

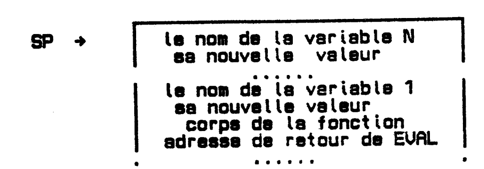
```
EVEXP:
    NOP     ,,[CALL (EVALN)] ; évalue les arguments
    POP     A4               ; A4 ← la F-val
    CDR     A4,(@ CORPS)     ; sauvée dans CORPS
    PUSH    ':MARK:          ; marque la pile
    CAR     A4,A2,[JUMP (BIND2)] ; A2 ← <svar>
BIND1:
    CAR     A2,A3            ; A3 ← variable suivante
    CVAL    A3,TST           ; sauve son ancien. C-val
    CAR     A1,A4            ; A4 ← valeur suivante
    SCVAL   A4,A3            ; nouvelle C-val
    CDR     A2,A2            ; avance dans les var.
    CDR     A1,A1            ; avance dans les val.
BIND2:
    TLIST   A2,,[JUMP (BIND1)] ; variable suivante
    TNIL    A2,,[JUMP (BIND3)] ; terminé
    CVAL    A2,TST           ; dernière valeur
    PUSH    A2               ; dernier nom
    SCVAL   A1,A2            ; dernière liaison
BIND3:
    MOVE    (@ CORPS),A1     ; A1 ← corps
```

Cette méthode facile à décrire et à comprendre présente l'énorme défaut de consommer des doublets de liste indirectement par le sous-programme EVALN, augmentant d'autant le nombre de récupérations de la zone liste. De plus tous les arguments sont évalués, même ceux en surnombre, qui ne seront liés à aucune variable.

La deuxième méthode utilisée pour réaliser cette liaison va permettre de ne consommer que des ressources de type pile. Pour assurer un parallélisme dans la réalisation des liaisons, il faut évaluer TOUTES les valeurs des arguments AVANT de réaliser les liaisons proprement dites qui vont affecter les *C-VAL* des variables.

Ceci est réalisé en utilisant un algorithme comportant deux phases :

1) évaluation des arguments et fabrication dans la pile d'un bloc possédant la structure suivante :



2) réalisation des liaisons proprement dites, en échangeant les nouvelles valeurs des variables dans la pile avec leurs anciennes *C-VAL*.

Voici le code de ces deux phases :

```
EVEXP:
    POP     A4                      ; A4 ← la F-VAL
    CAR     A4,A2                   ; A2 ← <svar>
    CDR     A4,TST,[JUMP (EVEXP2)] ; sauve le corps

; Fabrication des emplacements  dans la pile
; et évaluation des arguments

EVEXP1:
    CDR     A1,TST                  ; sauve le reste des val.
    PUSH    A2,,[CALL (EVAL1)] ; empile les variables
    POP     A2                      ; récupère  les variables
    XTOPST  A1                      ; force la valeur évaluée
    CAR     A2,TST                  ; empile le nom
    CDR     A2,A2                   ; variable suivante
EVEXP2:
    TLIST   A2,,[JUMP (EVEXP1)] ; il en reste
    TNIL    A2,,[JUMP (EVEXP3)] ; c'est la fin
    PUSH    A2,,[CALL (EVALN)]  ; évalue la liste
    XTOPST  A1                      ; empile la valeur
    PUSH    A1                      ; empile le nom

; Réalise la liaison superficielle

EVEXP3:
    STACK   A4,,[JUMP (EVEXP5)] ; garde la hauteur
EVEXP4:
    CVAL    A2,A1                   ; récupère la C-VAL
    XTOPST  A1                      ; qui est sauvée
    SCVAL   A1,A2                   ; nouvelle liaison
    POP     A1                      ; saute la valeur
EVEXP5:
    POP     A2                      ; nouvelle variable
    TATOM   A2,,[JUMP (EVEXP4)] ; il y en a encore
    PUSH    ':MARK:'                ; marque la pile

EVEXPN:                             ; appel normal
    SSTACK  A4                      ; tête de bloc
    MOVE    A2,A1                   ; A1 ← le corps
```

C'est cette méthode qui est utilisée dans notre modèle.

Une fois les variables liées, il faut exécuter le corps de la fonction puis délier les variables.

```
                ; Exécution du corps de la fonction

{140}  EXEC:
{141}     NOP    ,,[CALL (PROGN)] ; évalue le corps

                ; Délie les variables

{142}  UNBIND:
{143}     POP    A4                      ; dépile encore
{144}     EQ     A4,'¤MARK¤,[RETURN]     ; c'est fini
{145}     SCVAL  TST,A4,[JUMP (UNBIND)]; retour
```

{140} EXEC : est l'étiquette du sous-programme qui évalue le corps de la fonction. Ce corps doit être contenu dans A1.

{141} après avoir chargé dans A1 le corps de la fonction (i.e. le CDR de la *F-VAL*) *{103, 130, 135}*, il faut appeler le sous-programme PROGN, qui va évaluer les différents éléments de la liste A1 et retourner en valeur la valeur de la dernière évaluation.

Ce sous-programme s'écrit :

```
PROGN: CDR   A1,A2
       FLIST A2,,[JUMP (EVAL1)]
       PUSH  A2,,[CALL (EVAL1)]
       POP   A1,,[JUMP (PROGN)]
```

{142} au retour du PROGN, A1 contient la valeur de la fonction et le sommet de la pile a la forme :

```
SP  →     | nom de la variable N  |
          | son ancienne C-VAL    |
                    ...
                    ...
          | nom de la variable 1  |
          | son ancienne C-VAL    |
          |        ¤MARK¤         |
          |   adresse de retour   |
                    ...
```

Il faut donc délier les variables, en dépilant successivement tous les couples variable-valeurs sauvés dans la pile, jusqu'à la rencontre du marqueur de pile ¤MARK¤, ce qui est réalisé par *{143}*, *{144}* et *{145}*.

4.6.4 L'évaluation des fonctions de type FEXPR et MACRO

L'évaluation des fonctions de type FEXPR ne se différencie de l'évaluation des EXPR que par la liaison des arguments. Les arguments ne devant pas être évalués, la fabrication du bloc de sauvegarde dans la pile ne pose pas les problèmes soulevés par les EXPR.

Evaluation des FEXPR

```
{154} EVFEXP:
{155}     POP     A4                      ; A4 ← la F-VAL empilée
{156} EVFEXB:                             ; commun FEXPR/MACRO
{157}     CAR     A4,A2                   ; A2 ← liste de variables
{158}     PUSH    '¤MARK¤,,[JUMP (EVFEX3)]
{159} EVFEX2:
{160}     CAR     A2,A3                   ; A3 ← variable suivante
{161}     CVAL    A3,TST                  ; sauve l'ancienne CVAL
{162}     SCVAL   A1,A3                   ; force la nouvelle val.
{163}     PUSH    A3                      ; sauve le nom de la var.
{164}     MOVE    NIL,A1                  ; autres valeurs à NIL
{165}     CDR     A2,A2
{166} EVFEX3:
{167}     TLIST   A2,,[JUMP (EVFEX2)]     ; il reste des var.
{168}     CDR     A4,A1,[JUMP (EXEC)]     ; exécute le corps
```

Evaluation des MACRO

```
{150} EVMAC:
{151}     MOVE    (@ FORME),A1
{152}     POP     A4
{153}     PUSH    (EVAL),,[JUMP (EVFEXB)]
```

{150} est l'adresse de la routine spécialisée dans l'évaluation des MACRO.

{151} prépare dans A1 l'argument de la MACRO qui est par définition la forme elle-même, sauvée dans FORME en *{2}*.

{152} libère la pile en chargeant dans A4 la *F-VAL* de la MACRO qui avait été empilée en *{17}*.

{153} permet de réaliser la double évaluation en utilisant la propriété de la pile de contrôle :

```
NOP     ,,[CALL (EXFEXB)]
NOP     ,,[JUMP (EVAL)]
```

équivalent à :

```
PUSH    (EVAL),,[CALL (EVFEXB)]
```

4.6.5 <u>L'évaluation des fonctions numériques</u>

Dans le cas où la fonction est un nombre **<n>**, elle retourne le **<n>**ième élément du 1er argument évalué qui doit être une liste. Il y a donc un appel impicite de la fonction VLISP CNTH.

Exemple : (3 '(A B C D E)) ☞ C

Cette propriété permet d'accéder directement à n'importe quel élément d'une liste sans avoir à manier de longues chaînes de sélecteurs CAR/CDR.

(9 <l>) est équivalent à (CAR (CDDR (CDDDR (CDDDR <l>))))

Voici le code VCMC2 traitant des fonctions numériques :

<u>Evaluation des fonctions numériques</u>

```
{170} EVALFN:
{170}    PUSH   A2,,[CALL (EVAL1)]        ; A1 ← la val de la liste
{171}    POP    A2,,[JUMP (EVALN2)]       ; A2 ← le nb
{172} EVALN1:
{173}    CDR    A1,A1                     ; avance dans la liste
{174}    FLIST  A1,,[RETURN]              ; si la liste est vide
{175} EVALN2:
{176}    SUB    '1,A2
{177}    GT     A2,'0,[JUMP (EVALN1)]     ; il faut encore avancer
{178}    CAR    A1,A1,[RETURN]            ; ramène l'élément pointé
```

4.6.6 L'évaluation des fonctions composées

Si la fonction n'est ni un symbole atomique, ni un nombre, il s'agit alors d'une fonction composée.

Cette fonction est une liste qui peut être la description anonyme de la fonction ou une nouvelle forme dont la valeur sera la fonction. Il s'agit de fonctions calculées.

Les fonctions anonymes possèdent deux formes :
- la forme λ
- la forme INTERNAL

Ces deux formes sont détectées par tests du 1er élément de la liste, et sont évaluées d'une manière analogue aux fonctions associées aux symboles atomiques.
En revanche les fonctions calculées demandent une nouvelle évaluation dont la valeur est la fonction. En terme de syntaxe, VLISP utilise donc l'appel par nom des fonctions. L'utilisation de ces fonctions calculées, permet une très grande souplesse quant à l'utilisation des fonctions.

Voici le code VCMC2 traitant des fonctions composées :

Evaluation des fonctions composées

```
EVALFS:

    CAR     A2,A3                     ; A3 ← CAR de la fonct.
    EQ      A3,'LAMBDA,[JUMP (EVALL)] ; c'est une λ
    EQ      A3,'INTERNAL,[JUMP (EVALI)] ; fonction INTERNAL
    PUSH    A1                        ; sauve les arguments
    MOVE    A2,A1,[CALL (EVAL)]       ; évalue la fonction
    MOVE    A1,A2                     ; pour EVALFU
    POP     A1,,[JUMP (EVALFU)]       ; re-évalue la forme

EVALL:                   ;---    la fonction est une λ

    CDR     A2,TST,[JUMP (EVEXP)]

EVALI:                   ;---    la fonction est INTERNAL

    CDR     A2,A2                     ; A2 ← (FTYP FVAL)
    CAR     A2,A3                     ; A3 ← FTYP
    CDR     A2,A2                     ; A2 ← (FVAL)
    CAR     A2,TST,[JUMP (EVALIN)];   pile ← FVAL
```

4.7 COMMENT DEFINIR DES FONCTIONS EN VLISP ?

Définir une fonction c'est associer à un symbole atomique un couple ***F-TYPE, F-VAL***. Cette association va changer les propriétés naturelles ***F-TYPE*** et ***F-VAL*** de l'atome.

On peut définir de nouvelles fonctions de deux manières :
- statiquement
- dynamiquement

Toute fonction définie statiquement restera définie jusqu'à ce qu'une nouvelle définition lui soit donnée explicitement. La durée de vie d'une telle fonction est entièrement contrôlée par l'utilisateur.

Une fonction définie dynamiquement en revanche ne restera définie que le temps de l'évaluation d'une suite d'expressions fournie au moment même de la définition de la fonction. Ces définitions plus complexes seront décrites au chapitre 7.

VLISP permet de modifier statiquement les propriétés naturelles ***F-TYPE*** et ***F-VAL*** d'un atome de quatre manières distinctes :

1) en utilisant directement les fonctions d'accès aux propriétés naturelles des atomes. Il existe en effet 2 fonctions :

```
(FTYPE <a> <v>)
(FVAL  <a> <v>)
```

qui vont permettre d'accéder aux propriétés naturelles de l'atome <a>. Si le 2ème argument <v>, n'est pas fourni, l'accès est en lecture seule, en revanche, s'il est fourni, il devient la nouvelle valeur de la propriété.

2) en utilisant les fonctions de définition statique de type DE/DF/DM. VLISP possède en effet trois fonctions de définition statique :

```
(DE <a> <svar> <e1> ... <eN>)
(DF <a> <svar> <e1> ... <eN>)
(DM <a> <svar> <e1> ... <eN>)
```

qui permettent de declarer des définitions de type EXPR, FEXPR ou MACRO respectivement. Ces fonctions détruisent irrémédiablement

les anciennes propriétés ***F-VAL*** et ***F-TYPE*** des atomes concernés.

3) en utilisant les fonctions de re-définition statique de type RDE, RDF ou RDM. Ces fonctions (qui possèdent la même syntaxe que les fonctions précédentes) vont sauver les anciennes valeurs des propriétés avant de les modifier. Cette sauvegarde s'effectue par mise sur la ***P-LIST*** de l'atome d'une liste de la forme (<ftyp> <fval>) sous l'indicateur INTERNAL. Ces fonctions sont en général utilisées en phase de mise-au-point pour redéfinir statiquement des fonctions standards.

```
(RDE <a> <svar> <e1> ... <eN>)
(RDF <a> <svar> <e1> ... <eN>)
(RDM <a> <svar> <e1> ... <eN>)
```

Cette redéfinition est effectuée comme suit :

```
(ADDPROP <a> [ (FTYPE <a>) (FVAL <a>) ] INTERNAL)

(DE/DF/DM <a> <svar> <e1> ... <eN>)
```

4) en utilisant la fonction de restauration des anciennes valeurs des propriétés. Cette fonction :



permet de restaurer les propriétés naturelles de l'atome <a> précédemment sauvées au moyen d'une fonction du type précédent. Cette restauration est effectuée comme suit :

```
(FTYPE <a> (CAR (GET <a> INTERNAL)))
(FVAL <a> (CADR (GET <a> INTERNAL)))
(REMPROP <a> INTERNAL)
```

4.8 LES FONCTIONS STANDARDS.

La description de l'interprète de base est donc terminée. Toutefois un interprète VLISP ne peut pas fonctionner sans un certain nombre de fonctions standards (de type SUBR et FSUBR). Ces SUBR seront :

- des fonctions de contrôle
- des fonctions de manipulation d'objets VLISP
- des fonctions de test

Le texte de 160 fonctions standards (y compris les fonctions d'entrée/sortie) est donné dans les appendices **C**, **E** et **F**.

1	'	0SUBR	2	ABS	1SUBR
3	ADD1	1SUBR	4	ADDPROP	3SUBR
5	AND	FSUBR	6	APPEND	2SUBR
7	APPEND1	2SUBR	8	APPLY	2SUBR
9	APPLYN	NSUBR	10	ASCII	1SUBR
11	ASSQ	2SUBR	12	ATOM	1SUBR
13	BOUNDP	1SUBR	14	CAAAR	1SUBR
15	CAADR	1SUBR	16	CAAR	1SUBR
17	CADAR	1SUBR	18	CADDR	1SUBR
19	CADR	1SUBR	20	CALL	NSUBR
21	CAR	1SUBR	22	CASCII	1SUBR
23	CASSQ	2SUBR	24	CDAAR	1SUBR
25	CDADR	1SUBR	26	CDAR	1SUBR
27	CDDAR	1SUBR	28	CDDDR	1SUBR
29	CDDR	1SUBR	30	CDR	1SUBR
31	CHRONOLOGY	1SUBR	32	CNTH	2SUBR
33	COMPL	1SUBR	34	COND	FSUBR
35	CONS	2SUBR	36	COPY	1SUBR
37	CVAL	1SUBR	38	DE	FSUBR
39	DELQ	2SUBR	40	DF	FSUBR
41	DIFFER	2SUBR	42	DM	FSUBR
43	DMC	FSUBR	44	DMP	FSUBR
45	EOL	0SUBR	46	EPROGN	1SUBR
47	EQ	2SUBR	48	EQUAL	2SUBR
49	ERROR	NSUBR	50	ESCAPE	FSUBR
51	EVAL	3SUBR	52	EVLIS	1SUBR
53	EVLIS*	1SUBR	54	EXIT	FSUBR
55	EXITCHRONOL	FSUBR	56	EXPLODE	1SUBR
57	FINDCHRONOL	FSUBR	58	FREVERSE	2SUBR
59	FREVERSE*	2SUBR	60	FTYPE	2SUBR
61	FVAL	2SUBR	62	GE	2SUBR
63	GET	2SUBR	64	GT	2SUBR
65	IF	FSUBR	66	IFN	FSUBR
67	IMPLODE	1SUBR	68	INTERNAL	FSUBR
69	KWOTE	1SUBR	70	LAMBDA	FSUBR
71	LAST	1SUBR	72	LE	2SUBR
73	LENGTH	1SUBR	74	LESCAPE	FSUBR
75	LET	FSUBR	76	LETF	FSUBR
77	LIST	FSUBR	78	LIST*	FSUBR
79	LISTP	1SUBR	80	LMARGIN	1SUBR

81	LOGAND	2SUBR	82	LOGOR	2SUBR
83	LOGXOR	2SUBR	84	LT	2SUBR
85	MAP	NSUBR	86	MAPC	NSUBR
87	MCONS	FSUBR	88	MEMBER	2SUBR
89	MEMORY	2SUBR	90	MEMQ	2SUBR
91	MINUS	1SUBR	92	MINUSP	1SUBR
93	NCONC	2SUBR	94	NCONC1	2SUBR
95	NCONS	1SUBR	96	NEQ	2SUBR
97	NEQUAL	2SUBR	98	NEROP	1SUBR
99	NEWL	FSUBR	100	NEXTL	FSUBR
101	NOT	1SUBR	102	NTH	2SUBR
103	NULL	1SUBR	104	NUMBP	1SUBR
105	OR	FSUBR	106	OUTBUF	2SUBR
107	OUTPOS	1SUBR	108	PEEKCH	0SUBR
109	PLENGTH	1SUBR	110	PLIST	2SUBR
111	PLUS	2SUBR	112	PPRIN	1SUBR
113	PPRINT	1SUBR	114	PRETTY	FSUBR
115	PRIN	FSUBR	116	PRINCH	2SUBR
117	PRINT	FSUBR	118	PRINTLENGTH	1SUBR
119	PRINTLEVEL	1SUBR	120	PRINTLINE	1SUBR
121	PROG1	FSUBR	122	PROGN	FSUBR
123	PRSTACK	1SUBR	124	PTYPE	2SUBR
125	PUT	3SUBR	126	QUO	2SUBR
127	QUOTE	FSUBR	128	RDE	FSUBR
129	RDF	FSUBR	130	RDM	FSUBR
131	READ	0SUBR	132	READCH	0SUBR
133	REM	2SUBR	134	REMPROP	2SUBR
135	REVERSE	2SUBR	136	REVERT	1SUBR
137	RMARGIN	1SUBR	138	RPLACA	2SUBR
139	RPLACB	2SUBR	140	RPLACD	2SUBR
141	SELECTQ	FSUBR	142	SELF	FSUBR
143	SET	2SUBR	144	SETQ	FSUBR
145	SETQQ	FSUBR	146	STEPEVAL	1SUBR
147	STOP	0SUBR	148	SUB1	1SUBR
149	SUBST	3SUBR	150	SUBST*	3SUBR
151	SYNONYM	2SUBR	152	TERPRI	1SUBR
153	TIMES	2SUBR	154	TOPLEVEL	0SUBR
155	TYPECH	2SUBR	156	UNTIL	FSUBR
157	WHERE	FSUBR	158	WHILE	FSUBR
159	XCONS	2SUBR	160	ZEROP	1SUBR

De la puissance de ces fonctions standards dépend la puissance du système.

Nous considérerons donc que la fonction EVAL est l'<u>Operating System</u> d'un ensemble de fonctions élémentaires.

CHAPITRE 5

LES ENTREES

L'interprète de base vu au chapitre précédent ne possède pas de fonctions d'entrée/sortie. Nous décrirons donc dans ce chapitre, ainsi que dans le chapitre suivant toutes les fonctions standards d'entré/sortie de notre modèle d'implémentation.

VLISP étant très utilisé pour réaliser des traitements symboliques, les fonction d'entrée doivent être capables de lire aussi bien des expressions VLISP que d'autres textes symboliques.

5.1 LA SYNTAXE DES EXPRESSIONS VLISP

La syntaxe des expressions VLISP est extrêmement simple et découle de la syntaxe LISP utilisée de nos jours, la syntaxe des *S-expressions*. Originellement cette syntaxe n'avait été conçue que pour décrire les structures de données de LISP et non pas les programmes LISP qui disposaient d'une autre syntaxe, la syntaxe des *M-expressions*. Par la suite, seule la syntaxe des *S-expressions* a été utilisée. Toutes les tentatives d'utilisation d'une autre syntaxe en général apparentée à l'ALGOL (tels MLISP [SMITH 73], CLISP [TEITELMAN 73], CGOL [PRATT 76]) sont restées vaines.

Nous préférons à la place d'imposer une nouvelle syntaxe (dont l'utilité reste encore entièrement à démontrer) donner à l'utilisateur toutes les facilités pour contrôler lui-même voire totalement changer l'analyse lexicale et syntaxique des fonctions d'entrée.

La syntaxe VLISP se différencie de la syntaxe LISP classique par les points suivants :

1) il existe deux manières d'inclure des caractères spéciaux dans le nom externe (***P-NAME***) d'un symbole atomique :
   - en ***quotant*** chaque caractère spécial par le caractère ***quote-caractère***. Ce caractère est en général le **/** mais peut être redéfini.
   - en encadrant tous les caractères du ***P-NAME*** par le caractère délimiteur de ***P-NAME***. Ce caractère est en général le guillemet " mais peut être redéfini. Les symboles atomiques créés en utilisant cette 2ème méthode sont considérés comme des <u>constantes</u> i.e. que la valeur du symbole à sa création n'est pas la valeur indéfinie mais le symbole lui-même. Il n'est donc pas nécessaire en VLISP de quoter ces symboles atomiques pour les utiliser sous la forme de constantes. Ces symboles atomiques font ainsi office de pseudo-chaînes de caractères.

2) les expressions VLISP utilisent la forme la plus générale des S-expressions LISP :

   (<expression> . <expression>)

3) VLISP possède une nouvelle syntaxe pour décrire la construction dynamique de listes. Cette notation qui utilise des crochets carrés unifie en une seule description toutes les fonctions de création de listes :

l'expression VLISP	correspond à l'écriture LISP
[<e>]	(NCONS <e>) *{Note 1}*
[<e1> . <e2>]	(CONS <e1> <e2>)
[<e1> ... <eN-1> . <eN>]	(MCONS <e1> ... <eN>) *{Note 2}*
[<e1> ... <eN>]	(LIST <e1> ... <eN>)

{Note 1} cette fonction peut être définie en VLISP :

```
(DE NCONS (x) (CONS x NIL))
```

{Note 2} cette fonction peut être définie en VLISP :

```
(DF MCONS (L)
    (LET (L L)
        (CONS (EVAL (CAR L))
            (IF (NULL (CDDR L))
                (EVAL (CADR L))
                (SELF (CDR L))))))
```

l'écriture :

[[A . Y] [V] W X]

correspond à l'appel

(LIST (CONS A Y) (NCONS V) W X)

4) les commentaires sont encadrés du caractères début de commentaires (e.g. le ;) et du caractère fin de commentaires (e.g. le caractère fin de ligne). Le caractère début de commentaires peut être également employé comme caractère fin de commentaires ce qui permet d'avoir 2 types de commentaires :
   - les commentaires à l'intérieur d'une ligne
   - les commentaires jusqu'à la fin de la ligne.

5) Notre modèle possède des macro-caractères *{Note 1}* qui permettent d'accroitre considérablement les possibilités des fonctions standards de lecture.

   Le macro-caractère le plus fréquemment utilisé est le macro-caractère **QUOTE** qui produit le résultat suivant :

   '<s> est lu **(QUOTE <s>)**

{Note 1} un macro-caractère est un caractère auquel est associé une fonction VLISP. A la lecture de ce caractère par les fonctions d'entrée, la fonction associée est appelée automatiquement et la valeur de l'évaluation de cette fonction remplace le caractère qui avait été lu.

5.2 QUE PERMET LA MACHINE VCMC2 EN ENTREE

Trois instructions d'entrée sont suffisantes dans notre modèle :

READ	expression VLISP lue	→	<d>
IN	caractère lu	→	<d>
INTERN	forme interne de <s>	→	<d>

La première instruction **READ** n'est pas une véritable instruction VCMC2. En effet elle réalise la lecture complète d'une expression VLISP. Cette instruction n'est donc pas utilisée par les fonctions d'entrée dont le rôle est précisément de réaliser cette lecture mais sert à tester le fonctionnement de l'interprète indépendemment des fonctions d'entrée.

La seconde instruction **IN** est la seule intruction de lecture physique. Cette lecture a lieu dans un flux unique de caractères d'entrée. Nous n'aborderons pas dans cette étude les flux multiples d'entrée.

La dernière instruction **INTERN** est utilisée pour fabriquer la représentation interne d'un atome VLISP. La gestion de la zone allouée aux atomes dépend de l'incarnation propre de notre modèle. **INTERN** sert donc d'interface entre les modules de gestion de cet espace (allocation et recherche) et les modules d'analyses lexicales et syntaxiques d'entrée décrits en VCMC2.

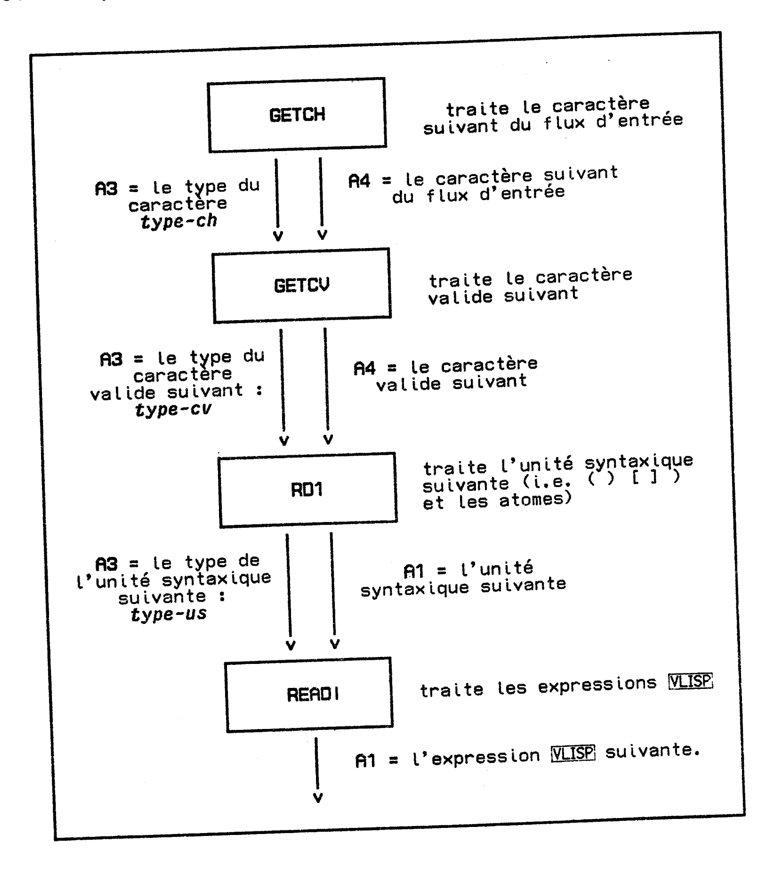
Classiquement cette gestion fait intervenir une liste des objets (i.e. de symboles atomiques) présents dans le système, l'**OBLIST**. Les symboles atomiques (de longeur variable (voir le paragraphe 2.2)) sont liés dans l'**OBLIST** au moyen d'un pointeur sur l'objet suivant, le ***A-LINK***, qui est une propriété naturelle de tout symbole atomique.

Ce *A-LINK* permet de réaliser à peu de frais un *hash-coding* de cette liste *{Note 1}*.

{Note 1} la clé du hash-coding utilisée par VLISP 10 est le reste de la division entière du nombre de caractères du P-NAME du symbole et de ses 4 premiers caractères (le tout sur 36 bits) par la valeur 71. Les résulats obtenus avec une **OBLIST** *toujours supérieure à 600 éléments est de 2,3 recherches par élément si on ajoute le raffinement supplémentaire suivant (induit par l'accès non-uniforme aux symboles) : tous les éléments trouvés sont systématiquement remis en tête de chaque sous-liste.*

5.3 L'ORGANISATION DES MODULES D'ENTREE

Nous avons découpé les fonctions d'entrée en plusieurs modules dont voici le synopsis. Le texte de ces modules est donné à l'appendice **C.**



Chacun de ces modules retournent 2 valeurs :
- l'objet traité par le module
- le type de cet objet.

Le retour systématique d'un type codé permet de réaliser toute l'analyse au moyen d'aiguillages.

1) le module **GETCH**
traite le caractère logique suivant. Il retourne dans **A4** le code interne de ce caractère et dans **A3** le type du caractère suivant ***type-ch.*** Note modèle possède à cet effet une table des types des caractères (la table **TABCH**) qui associe à chacun des caractères un type ***type-ch*** qui peut prendre les valeurs suivantes :

0	caractère à ignorer complétement
1	caractère début de commentaires (e.g. ;)
2	caractère fin de commentaires (e.g. RC)
3	caractère quote caractère (e.g. /)
4	caractère début de liste (e.g. ( )
5	caractère fin de liste (e.g.))
6	caractère début de liste évaluée (e.g. [ )
7	caractère fin de liste évaluée (e.g.])
8	caractère de paire pointée (e.g. .)
9	caractère séparateur (e.g. l'espace)
10	macro-caractère
11	caractère délimiteur de P-NAME (e.g. ")
12	caractère normal

2) le module **GETCV**
traite le caractère valide suivant. Ce module enlève les commentaires, réalise l'action du quote caractère, retourne le caractère valide suivant dans **A4** et le type de ce caractère (***type-cv***) dans **A3** parmi les types suivants :

0	caractère début de liste (e.g. ( )
1	caractère fin de liste (e.g.))
2	caractère début de liste évaluée (e.g. [ )
3	caractère fin de liste évaluée (e.g.])
4	caractère de paire pointée (e.g. .)
5	caractère séparateur (e.g. l'espace)
6	macro-caractère
7	caractère délimiteur de P-NAME (e.g. ")
8	caractère normal

3) le module **RD1** traite l'unité syntaxique suivante. Ce module traite les symboles atomiques, les pseudo-chaînes de caractères, appelle les fonctions associées aux macro-caractères, retourne dans **A1** l'objet VLISP suivant et dans **A3** le type de l'unité syntaxique (***type-us***) qui peut prendre les valeurs :

0	u.s. début de liste (e.g. ( )
1	u.s. fin de liste (e.g.))
2	u.s. début de liste évaluée (e.g. [ )
3	u.s. fin de liste évaluée (e.g.])
4	u.s. de paire pointée (e.g. .)
5	u.s. objet VLISP (valeur dans A1)

5.4 LES FONCTIONS STANDARDS D'ENTREE

Notre modèle permet d'utiliser ces modules et d'accéder à la table des type des caractères au moyen de fonctions standards.

Nous avons inclu 8 fonctions standards d'entrée dans notre modèle :
- 2 fonctions de lecture d'expressions VLISP,
- 2 fonctions de lecture de caractères,
- les fonctions de conversion code interne vers caractère,
- la fonction d'accès à la table de l'analyseur lexical
- et une fonction de définition de nouveaux macro-caracteres.

Ce nombre volontairement limité, va cependant permettre tous les types de lecture.

Les fonctions de lecture des expressions VLISP détectent les erreurs de syntaxe dans ces expressions. Ces erreurs sont des erreurs fatales et provoquent l'arrêt de la lecture.

Fonctions de lecture d'expressions VLISP :
(READ)
(IMPLODE <l>)

La fonction **READ** réalise la lecture dans le flux d'entrée de caractères tandis que la fonction **IMPLODE** possède un argument <l> qui est une liste de caractères faisant office de flux d'entrée. Cette fonction permet un accès direct à toutes les possibilités de création d'objets internes.

Ainsi l'appel :

```
(IMPLODE '(  "'" "(" A B " " C ")" ))
```

livrera en résultat la liste :

```
(QUOTE (AB C))
```

Il est également souhaitable pour réaliser des lectures particulières de lire le flux d'entrée caractère par caractère. Ceci est réalisé au moyen des deux fonctions :

Fonction de lecture de caractère :

(READCH)

(PEEKCH)

Ces deux fonctions retournent en valeur le caractère suivant sous la forme d'un symbole atomique mono-caractère et ne se différencient que par l'état du flux d'entrée aprés leur invocation : la fonction **READCH** fait disparaître le caractère du flux d'entrée tandis que la fonction **PEEKCH** ne fait que consulter le caractère et laisse le flux d'entrée inchangé.

Notre système permet également de convertir des caractères en leur équivalent interne.

Fonctions de conversion

(ASCII <n>)

(CASCII <c>)

La première fonction **ASCII** convertit le code interne **<n>** en son équivalent caractère et la seconde **CASCII** retourne le code interne de son argument qui doit être un symbole atomique mono-caractère.

Enfin il existe une fonction d'accès à la table des types des caractères.

Fonction d'accès à la table des types

(TYPECH <c>) lecture
(TYPECH <c> <n>) écriture

Cette fonction permet donc de connaître ou de modifier tout ou partie de la table des types des caractères d'entrée (la table **TABCH**) facilitant ainsi la relecture des programmes LISP écrits dans d'autres dialectes LISP. La valeur **<n>** est un type de caractère (***type-ch***) tel

qu'il est donné au paragraphe 5.3

Enfin la dernière fonction standard d'entrée est la fonction de définition de nouveaux macro-caractères. La syntaxe de cette fonction est identique à celle de toutes les autres fonctions de définition.

Fonction de définition d'un nouveau macro-caractère

(DMC <c> <sarg> <e1. ... <eN>)

qui peut être défini en VLISP :

```
(DF DMC (l)
   (TYPECH (CAR l) 10)
   (EVAL ['DE . l]))
```

CHAPITRE 6
LES SORTIES

Les possibilités de sortie de notre modèle d'implémentation doivent recouvrir tant les éditions automatiques des expressions VLISP avec ou sans formatage que les éditions de résultats qui doivent être entièrement contrôlées par l'utilisateur.

Il existe deux grands types d'impression des expressions VLISP :
- les impressions destinées à être relues par les utilisateurs
- les impressions destinées à être relues par le système (i.e. par la fonction READ).

Les problèmes ne sont évidemment pas les mêmes. Les impressions destinées aux utilisateurs se doivent d'être très lisibles, aérées, faisant resortir la structure de contrôle des expressions VLISP et ne doivent pas respecter à la lettre la syntaxe d'entrée de la fonction READ. Ces impressions vont faire appel à des techniques de typographie classique utilisées dans les systèmes de composition automatiques (appelés parfois *document compilers*) tels le système RUNOFF [DEC 75b], le système PUB [TESLER 72], le système POX [MAAS 78] ou le système TEX [KNUTH 78a, 78b]. En revanche les impressions destinées à être relues par le système doivent répondre exactement à la syntaxe d'entrée de la fonction READ et être si possible très compactes pour minimiser la place occupée pour le stockage des expressions et par conséquent les temps de lecture. Ces deux types d'impression sont incompatibles et c'est pourquoi notre modèle possède deux familles de fonctions d'impression.

En plus du problème d'impression des expressions VLISP le système de sortie va donner accès à tous les objets internes utilisés par les différents modules de sortie pour pouvoir redéfinir entièrement un système de sortie sur mesure.

Le texte des fonctions de sortie du système, écrites en VCMC2, est donné à l'appendice E ainsi que celui d'un formateur d'expressions VLISP, écrit en VLISP, à l'appendice D.

6.1 QUE PERMET DE FAIRE LA MACHINE *VCMC2* EN SORTIE ?

La machine VCMC2 possède un nombre très restreint d'instructions spéciales de sortie qui se composent d'une instruction de sortie physique et de deux instructions manipulant les représentations externes des atomes, i.e. manipulant les *P-NAME* des atomes (voir le chapitre 2).

La machine VCMC2 possède une instruction qui réalise les sorties physiques. Ces sorties physiques ont toutes lieu dans un <u>flux unique de sortie de caractères</u> (cette étude ne traitera pas des flux multiples en sortie).
Cette instruction envoie un caractère dans le flux de sortie. Son opérande source <s> contient le code interne du caractère à transmettre.

Voici la syntaxe de l'instruction de sortie physique :

OUT	<s> → flux de sortie

Les 2 instructions suivantes manipulent la représentation externe des atomes :

PLEN	(PLENGTH <s>) → <d>
PNAM	(PNAM <s>) → <d>

PLEN transfère dans l'opérande destination <d> la longueur de la représentation externe de l'atome contenu dans l'opérande source (i.e. le nombre de caractères de son ***P-NAME***). Cette instruction correspond à la fonction VLISP PLENGTH.

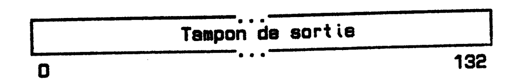
PNAM force les caractères de la représentation externe de l'atome contenu dans l'opérande source <s> dans des mots mémoire indexés par l'opérande destination <d> qui fait office de tampon.

Ces deux instructions ont été ajoutées à la machine de base VCMC2 pour pouvoir simuler les fonctions de sortie sans avoir à tenir compte des représentations physiques des atomes.

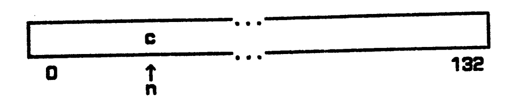
6.2 L'ORGANISATION ET L'UTILISATION DU TAMPON DE SORTIE.

Le système va utiliser dans les modules de sortie les objets internes suivants :
- un tampon de sortie
- un pointeur courant sur ce tampon
- et des marges gauche et droite sur ce tampon

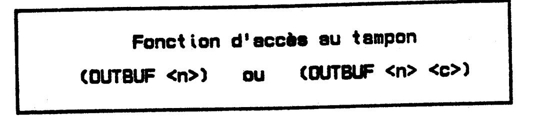
Le tampon de sortie est utilisé par VLISP pour composer ses lignes mais peut être utilisé directement au moyen de fonctions spécifiques.



Ce tampon fait office de "composteur" dans lequel VLISP va constituer une ligne à imprimer. Il est constitué de 133 emplacements contigus de la mémoire. Ce nombre est déterminé par la taille maximum d'une ligne d'imprimante. L'accès aux différents éléments de ce tampon est réalisé en utilisant le registre d'index de la machine VCMC2.

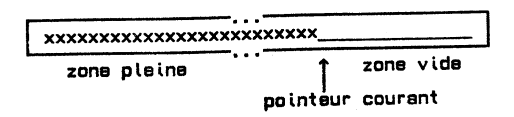


Ce tampon est disponible également en VLISP au moyen d'une fonction d'accès spéciale, la fonction **OUTBUF**, qui considère le tampon comme un vecteur. Le nom **OUTBUF** provient de **OUT**put **BUF**fer.



Cette fonction est de type 2SUBR. Son premier argument <n> (de type nombre) est l'indice de l'élément du tampon à considérer (le premier élément possède l'indice 0). Si le deuxième argument n'est pas fourni, **OUTBUF** retourne le contenu actuel de l'élément du tampon sous forme d'un caractère. Si le deuxième argument <c> est fourni, il remplace l'ancien contenu de l'élément du tampon et **OUTBUF** retourne cette nouvelle valeur

Le système tient continuellement à jour un pointeur sur ce tampon. Ce pointeur contient à tout moment la position du premier emplacement libre du tampon.

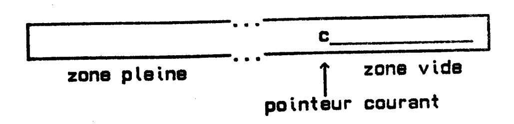


Comme pour le contenu du tampon lui-même, il est possible d'accéder à ce pointeur au moyen d'une fonction VLISP spécialisée, la fonction **OUTPOS**. Le nom **OUTPOS** provient de **OUT**put buffer **POS**ition

Fonction d'accès au pointeur courant

(OUTPOS) ou (OUTPOS <n>)

Cette fonction est de type 1SUBR. Si l'argument <n> (de type nombre) est fourni, il devient la nouvelle valeur du pointeur courant sur le tampon de sortie. Dans les deux cas, OUTPOS retourne la valeur actuelle du pointeur courant sur le tampon de sortie.



Le pointeur courant autorise une autre type de chargement du tampon en utilisant implicitement sa position. La fonction **PRINCH** réalise le chargement ***à la suite*** de ce qui se trouve déjà dans le tampon.

Fonction d'édition de caractères

(PRINCH <c>) ou **(PRINCH <c> <n>)**

Cette fonction, de type 2SUBR, va charger dans le tampon de sortie <n>

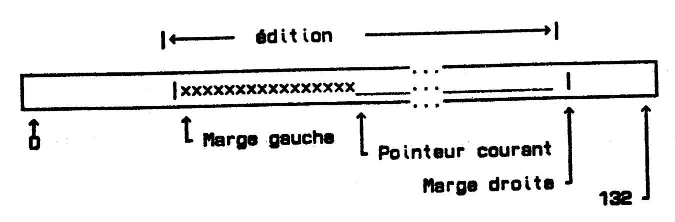
fois (ou 1 seule fois si le deuxième argument <n> n'est pas fourni) le caractère **<c>**. Après chaque chargement dans le tampon, le pointeur courant est actualisé. Si après un chargement le tampon devient plein, il est automatiquement vidé (au moyen de la fonction suivante **TERPRI**) et le pointeur courant est reinitialisé en début de tampon.

Une fois le tampon chargé au moyen des fonctions **OUTBUF** ou **PRINCH**, il faut pouvoir envoyer tous les caractères du tampon dans le flux de sortie. C'est l'objet de la fonction **TERPRI** dont le nom provient de **TER**minate **PRI**nt (ce qui indique bien que ce n'est qu'à l'exécution de cette fonction qu'il y a vraiment une sortie physique).

Fonction de vidage du tampon **(TERPRI) ou (TERPRI n)**

Cette fonction, de type **1SUBR**, réalise les actions suivantes :
- sortie physique (au moyen de l'instruction **OUT**) de tous les caractères contenus dans le tampon, de la position 0 (i.e. du début du tampon) jusqu'à la position du pointeur courant.
- sortie **<n>** fois (ou 1 seule fois si l'argument **<n>** n'est pas fourni) du caractère fin de ligne (au moyen de l'instruction **OUT**)
- effacement de tout le tampon
- positionnement du pointeur courant à la position 0 (i.e. au début du tampon)

Enfin le système a besoin de marges d'édition sur le tampon pour pouvoir décentrer toutes les impressions.



Le chargement du tampon ne s'opérera qu'entre ces deux marges. Ces marges sont positionnées au moyen de deux nouvelles fonctions VLISP, **LMARGIN** pour la marge gauche et **RMARGIN** pour la marge droite.

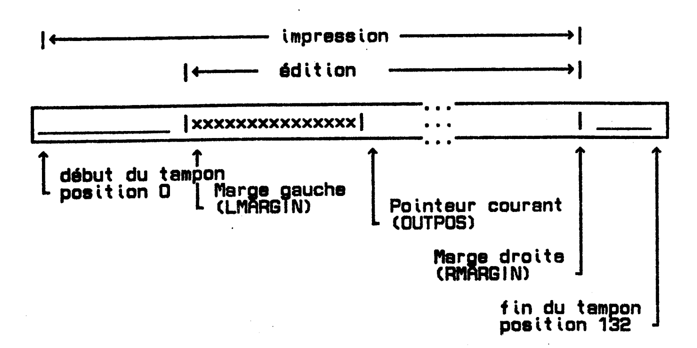
Fonctions de positionnement des marges

(LMARGIN) ou (LMARGIN <n>) et
(RMARGIN) ou (RMARGIN <n>)

Ces deux fonctions, de type 1SUBR, vont pouvoir initialiser les valeurs des marges. Par défaut elles sont de 0 pour la marge gauche et de 72 (qui est la taille standard d'une ligne sur un terminal) pour la marge droite. Ces fonctions retournent la valeur de la marge après modification. Si l'argument <n> n'est pas un nombre, la valeur de la marge n'est pas modifiée, mais la fonction retourne tout de même la valeur courante de la marge.

L'appel **(LMARGIN (+ (LMARGIN) 3))** avance donc la marge de gauche de trois positions.

Le tampon de sortie est donc organisé de la manière suivante :



Certaines fonctions commme **OUTPOS**, **OUTBUF**, **LMARGIN** ou **RMARGIN** testent si les positions fournies sur le tampon sont correctes. Au cas où elles ne le seraient pas un message d'avertissement est imprimé et la position courante est utilisée pour terminer la fonction.

Le message d'avertissement est de la forme :

***** Débordement de ligne :*** suivi de l'argument incorrect.

6.3 COMMENT SONT EDITES LES OBJETS USUELS VLISP.

Les listes sont éditées en utilisant la même notation qu'en entrée, (les éléments d'une liste seront séparés par des espaces et encadrés de parenthèses) mais où faut-il mettre des espaces ?
L'insertion d'un espace entre les différentes unités syntaxiques (atomes, parenthèses ouvrantes et fermantes et points) produit un bien étrange résultat :

en effet l'expression VLISP :

```
(DE FOO (L) (COND ((NULL L) ()) (T ...
```

serait éditée :

```
(_DE_FOO_(_L_)_(_COND_(_(_NULL_L_)_(_)_)_(_T ...
```

Nous utiliserons donc les règles d'espacements suivantes :

élément précédent	élément suivant: atome	(	)	.
atome	oui	oui	non	oui
(	non	non	non	/
)	oui	oui	non	oui
.	oui	/	/	/

l'expression VLISP :

```
(DE FOO (L) (COND ((NULL L) ()) ( ...
```

est donc éditée :

```
(DE_FOO_(L)_(COND_((NULL_L)_())_( ...
```

6.4 LES REGLES DE COMPOSITION SIMPLE DES EXPRESSIONS VLISP.

On peut imaginer plusieurs modes de composition des S-expressions :

1) La plus simple consiste à éditer dans le tampon de sortie tous les caractères de la représentation de l'expression VLISP. Au débordement du tampon celui-ci est vidé au moyen de la fonction **TERPRI** et l'édition se poursuit sur la ligne suivante. Le grand défaut de cette composition est qu'un atome peut être édité sur deux lignes, ce qui n'est ni esthétique ni facile à lire et empêche totalement la relecture par la fonction READ qui considère le changement de ligne comme un séparateur. Cette composition **en bloc**, très facile à réaliser, n'est donc jamais employée en VLISP.

exemple de composition en bloc

```
(DE GETREE1 (ARBRE CHEMIN) (IF ARBRE (IF (EQ (C
AR CHEMIN) (QUOTE %)) (TERMINE (GET ARBRE (QUOT
E %))) (GETREE1 (GET ARBRE (CAR CHEMIN)) (CDR C
HEMIN)) (TERMINE NIL)))
```

2) La deuxième méthode pour éditer des expressions VLISP va s'apparenter à la composition dite "en drapeau" chez les typographes. Avant de charger un atome, on teste d'abord si celui-ci rentre en entier dans le tampon. S'il rentre, l'atome est édité dans le tampon et le pointeur courant mis à jour. S'il ne rentre pas, le tampon est préalablement vidé dans le flux de sortie (au moyen de la fonction **TERPRI**) avant édition de l'atome dans un nouveau tampon. Pour réaliser ce test il faut connaître outre la position actuelle du pointeur courant, mais également le nombre de caractères de la représentation externe de l'atome (i.e. de son ***P-NAME***). Comme il est fastidieux de recalculer à chaque édition d'un atome la taille de son ***P-NAME***, une nouvelle propriété naturelle a été introduite ; il s'agit du ***P-LEN*** (abréviation de ***P***name ***LEN***gth). Cette propriété contient la taille du ***P-NAME*** de l'atome.

exemple de composition en drapeau simple

```
(DE GETREE1 (ARBRE CHEMIN) (IF ARBRE (IF (EQ (
CAR CHEMIN) (QUOTE %)) (TERMINE (GET ARBRE (
QUOTE %))) (GETREE1 (GET ARBRE (CAR CHEMIN)) (
CDR CHEMIN))) (TERMINE NIL)))
```

3) Cette composition lisible par tous (utilisateurs et fonction READ) peut être encore améliorée :

- La première amélioration consiste à n'imprimer les parenthèses ouvrantes que si l'on est sur que l'atome suivant rentre également dans la ligne. Ainsi dans les appels de fonctions les parenthèses ouvrantes associées aux noms de fonctions seront collées aux noms.

- la deuxième amélioration de la composition en drapeau consiste à éditer tous les appels de la fonction **QUOTE** (i.e. les appels de type **(QUOTE <s>)**) sous la forme **'<s>**. Il faut toutefois prendre garde de n'éditer que les appels de la fonction **QUOTE** de cette manière et non pas de n'importe quelle liste dont le CAR est l'atome QUOTE. Pour cela il suffit de vérifier que le CAR de la liste à éditer est bien l'atome QUOTE et que le CDDR de cette même liste est bien NIL car la fonction QUOTE ne possède qu'un argument.

exemple de composition en drapeau améliorée

```
[(DE GETREE1 (ARBRE CHEMIN) (IF ARBRE (IF (EQ     ]
 (CAR CHEMIN) '%) (TERMINE (GET ARBRE '%))
 (GETREE1 (GET ARBRE (CAR CHEMIN)) (CDR CHEMIN))
[) (TERMINE NIL)))                                ]
```

C'est ce type de composition que vont utiliser les principales fonctions de sortie des expressions VLISP, les fonctions **PRIN** et **PRINT**.

Fonction d'édition simple
(PRIN <e1> ... <eN>)

Cette fonction, de type FSUBR, va évaluer les différentes expressions **<e1>** ... **<eN>** et les éditer les unes à la suite des autres dans le tampon de sortie. **PRIN** retourne en valeur la valeur du dernier argument édité. Les différentes éditions sont séparées entre elles par l'insertion d'un espace.

Fonction d'impression simple
(PRINT <e1> ... <eN>)

Cette fonction édite les valeurs des différentes expressions **<e1>** ... **<eN>** d'une manière identique à la fonction **PRIN** mais effectue également un vidage du tampon (en utilisant la fonction **TERPRI**) après édition de la dernière valeur. **PRINT** retourne la valeur de la dernière expression éditée.

l'appel **(PRINT <e1> ... <eN>)** est donc équivalent à
l'appel **(PROG1 (PRIN <e1> ... <eN>) (TERPRI))** .

6.5 LES REGLES DE COMPOSITION AVANCEE DES EXPRESSIONS VLISP.

La composition en drapeau des fonctions **PRIN** et **PRINT** est adéquate pour éditer des expressions VLISP mais se révèle insuffisante :

- si on ne désire voir qu'une partie de l'expression (comme dans le cas des traces) : il n'est pas possible d'avoir une édition abrégée (ou condensée),

- si on désire avoir une vision claire de la structure de contrôle de l'expression (comme dans le cas des fonctions) : les lignes composées ne permettent pas une compréhension rapide des structures de contrôle de la fonction (voir les exemples du paragraphe précédent).

Nous avons donc introduit dans notre modèle deux nouvelles familles de fonctions :
- les fonctions de limitation d'impression
- les fonctions de belle-composition.

6.5.1 Les limitations des impressions

Les limitations d'impressions peuvent agir sur 3 paramètres :
- le nombre de lignes imprimées
- le nombre d'éléments de chaque liste imprimée
- la profondeur d'impression de chaque liste imprimée.

A chacune de ces limitations est associée une fonction :

Fonctions de limitations

(PRINTLINE)	ou	(PRINTLINE <n>)
(PRINTLENGTH)	ou	(PRINTLENGTH <n>)
(PRINTLEVEL)	ou	(PRINTLEVEL <n>)

Ces trois fonctions permettent d'accéder tant en lecture qu'en écriture (en lecture si l'argument <n> n'est pas fourni, en écriture s'il l'est) aux trois paramètres qui permettent de limiter les impressions :

1) **PRINTLINE** contrôle le nombre de lignes imprimées. Si le nombre de lignes à imprimer est plus grand que <n>, l'impression du reste de l'expression est abandonnée.

2) **PRINTLENGTH** contrôle le nombre d'éléments de liste imprimés. Si le nombre d'éléments de liste imprimés est plus grand que <n>, l'impression s'arrête et les fonctions impriment des points de

suspension suivis d'une parenthèse fermante.

3) **PRINTLEVEL** contrôle la profondeur maximum d'impression (i.e. le nombre de parenthèses ouvrantes non encore refermées). Si ce nombre est atteint, les fonctions d'impression éditent le caractère spécial "&" et l'impression continue au même niveau.

Voici des exemples d'utilisation des fonctions de limitations d'impressions. D'autres exemples seront donnés au paragraphe 6.7.

```
? (SETQ L
?   (LET ((n 100) (l))
?      (IF (ZEROP n) l (SELF (SUB1 n) (CONS n l)))))
=  (1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
= 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
= 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62
= 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82
= 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100)

? (PRINTLINE 2)
= 2

? L
=  (1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
= 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42

? (PRINTLINE 1000)
= 1000

? (PRINTLENGTH 3)
= 3

? '(A B C D E F)
= (A B C ...)

? '(A (B C D E) F)
= (A (B C ...) ...)

? (PRINTLENGTH 1000)
= 1000

? (PRINTLEVEL 3)
= 3

? '(A (B (C (D (E) F) G) H) I)
= (A (B (C & G) H) I)

? (PRINLEVEL 1000)
= 1000
```

6.5.2 La belle composition (ou PRETTY-PRINT)

Le besoin d'avoir un type de composition approprié à l'édition des expressions VLISP de type fonction se fait sentir dès que les fonctions dépassent quelques lignes de composition en drapeau.
Ce type de composition (qui est appelé PRETTY-PRINT dans le jargon LISPien [GOLDSTEIN 73]) doit être à même de faire ressortir la structure de contrôle de la fonction.

Le moyen le plus employé consiste à utiliser des alignements et des renfoncements droits.
Dans ce type de composition l'édition d'un appel de fonction va consister à :
- éditer une parenthèse ouvrante
- éditer à la suite le nom de la fonction
- puis pour chacun des arguments changer de ligne en avançant la marge gauche de trois positions. Ces arguments sont édités récursivement.
- éditer une parenthèse fermante.

Voici en VLISP la fonction réalisant ce type d'édition :

```
(DE PP (s)
    (IF (ATOM s)
        (PRIN s)
        (PRINCH "(")
        (PP (NEXTL s))
        (LMARGIN (+ (LMARGIN) 3))
        (WHILE (LISTP s)
            (TERPRI)
            (PP (NEXTL s)))
        (LMARGIN (- (LMARGIN) 3))
        (PRINCH ")")))
```

et voici un exemple de composition utilisant cette fonction :

```
(DE
   GETREE1
   (ARBRE
      CHEMIN)
   (IF
      ARBRE
      (IF
         (EQ
            (CAR
               CHEMIN)
            (QUOTE
               |))
         (TERMINE
            (GET
               ARBRE
               .......
```

Cette composition très aérée permet de montrer la nécessité d'avoir en réalité plusieurs types de composition suivant la fonction à éditer. En effet il est clair qu'il faut disposer les trois premiers arguments de la fonction DE (qui correspondent respectivement à DE, suivi du nom de la fonction, suivi de la liste d'arguments) sur la même ligne et tous les arguments suivants sur des lignes différentes (correspondant aux différentes expressions du corps de la fonction), de même il est clair qu'il faut imprimer l'argument de la fonction CAR au même niveau que l'atome CAR lui-même (car cette fonction n'a jamais plus d'un argument).

Nous avons défini 6 modèles d'édition qui recouvrent toutes les éditions des fonctions standards.

Modèle 0 : dénote l'absence de modèle. L'appel de la fonction est édité en utilisant une compositon en drapeau améliorée vue précédemment.

Modèle 1 : c'est le modèle de type **PROGN**. Chaque argument de la fonction est édité sur une ligne séparée après renfoncement de la marge gauche de 3 positions.

modèle 1	exemple
(nom-de-la-fonction argument 1 ... argument N)	(PROGN (PRINCH " ") (PRINCH ".") (PRINCH " "))

Modèle 2 : c'est le modèle de type IF. Le premier argument est édité sur la même ligne que le nom de la fonction et tous les autres arguments sont édités un par ligne comme dans le modèle précédent.

modèle 2	exemple
(nom-de-la-fonction argument-1 argument-2 ... argument-N)	(IF (ZEROP n) 1 (= n 2))

Modèle 3 : c'est le modèle de type **DE**. Les 2 premiers arguments sont édités sur la même ligne que le nom de la fonction et tous les autres arguments sont édités un par ligne comme dans le modèle précédent.

```
modèle 3                                exemple

(nom argument-1 argument-2          (DE FOO (n1 n2)
   argument-3                           (PRINT n1 n2)
      ...                               (+ n1 n2))
   argument-N)
```

Modèle 4 : ce modèle n'est utilisé que pour éditer la fonction **COND** qui possède une syntaxe spéciale décrivant les clauses. Il faut éditer chacune des clauses dans le modèle du **PROGN** (Modèle 1).

```
   modèle 4

(COND
   clause-1          toutes les clauses
     ...             sont éditées en
   clause-N)         utilisant le modèle 1
```

Modèle 5 : c'est le modèle réservé de la fonction **SELECTQ**. Le premier argument (dans ce cas le selecteur) est édité sur la même ligne puis les autres arguments (i.e. les clauses) sont éditées de la même manière que dans le modèle 4 (celui de la fonction **COND**).

```
   modèle 5

(SELECTQ argument
   clause-1          toutes les clauses
     ...             sont éditées en
   clause-N)         utilisant le modèle 1
```

Modèle 6 : c'est le modèle réservé de la fonction **SETQ** multiple. Chaque couple d'arguments (i.e. chaque couple nom/valeur) est imprimé sur une ligne séparée.

```
          modèle 6

(SETQ
   argument-1   argument-2
          ...
   argument-N-1   argument-N)
```

Cet ensemble de modèles défini, il nous reste à voir comment associer aux atomes fonctions des modèles particuliers.
Les fonctions du PRETTY-PRINT étaient originellement écrites en VLISP (voir l'appendice D). La sélection du type de modèle s'opérait en effectuant une comparaison du nom de la fonction avec des listes de noms associés à chaque modèle (cette comparaison utilisait en général la fonction **MEMQ**). Cette solution très lente n'est pas envisageable pour un PRETTY-PRINT écrit en langage machine (si ce n'est par macro-génération de tous les tests du MEMQ).

Pour rendre le temps de recherche indépendant du nombre de fonctions utilisant le même modèle, chaque fonction possèdait sur sa P-liste un numéro de modèle d'édition sous un indicateur spécial (en général PRETTY). La recherche du numéro de modèle s'effectuait donc au moyen d'un **GET**, qui pour être en général plus rapide que le MEMQ n'en coutait pas moins de 2 doublets par modèle spécifié.

Pour résoudre le problème d'accès et de place allouée au numéro des modèles, nous avons ajouté une nouvelle propriété naturelle aux symboles atomiques, le *P-TYPE* (qui signifie Print-*TYPE*) qui va contenir le numéro du modèle à utiliser pour éditer ce symbole en tant que fonction.
L'accès au *P-TYPE* est immédiat (à une indexation près) et la place occupée est minimisée (dans la plupart des implantations ce *P-TYPE* occupe un octet).
Pour accéder à ce nouvel attribut atomique, nous avons introduit une nouvelle fonction, la fonction **PTYPE**.

Fonction d'accès au *P-TYPE*

(PTYPE <a>)   ou   (PTYPE <a> <n>)

Cette fonction, de type 2SUBR, permet d'accéder au *P-TYPE* du symbole atomique <a>. Si le 2ème argument numérique <n> est fourni, il devient le nouveau *P-TYPE* du symbole <a>. Dans tous les cas la fonction **PTYPE** retourne sous forme d'un nombre le *P-TYPE* actuel du symbole <a>.

Nous avons inclu dans notre modèle d'implémentation un ensemble de fonctions standards qui utilisent ces *P-TYPE*s et possédent tous les modèles de composition décrits ci-dessus. Le texte de ces fonctions est donné à l'appendice **E**.

Ces fonctions vont permettre d'éditer des expressions VLISP quelconques ou bien des *F-VAL* en utilisant la belle composition. Toutes les expressions VLISP complexes de cette étude ont été composées en utilisant ces fonctions.

Fonction d'édition avancée

(PPRIN <s>)

Cette fonction, de type 1SUBR, va éditer l'argument <s> dans le tampon de sortie. **PPRIN** retourne en valeur la valeur de l'argument.

Fonction d'impression avancée

(PPRINT <s>)

Cette fonction édite la valeur de l'expression <s> d'une manière identique à la fonction **PPRIN** mais effectue également un vidage du tampon (en utilisant la fonction **TERPRI**) après édition. **PPRINT** retourne également la valeur de l'expression éditée.

l'appel **(PPRINT <s>)** est donc équivalent à
l'appel **(PROG1 (PPRIN <s>) (TERPRI))**

6.6 LES MACRO D'EDITION.

Pour permettre de résoudre les problèmes de sorties spéciales, la fonction d'édition **PPRIN** n'est pas figée, mais bien au contraire ne contient que les options par défaut de l'édition. Notre modèle permet en effet d'associer à chaque symbole atomique une MACRO d'édition spécialisée qui est une fonction qui sera utilisée à la place du modèle d'édition standard.

Ces MACRO d'édition sont définies au moyen de la fonction de définition des MACRO d'édition, **DMP**.

Fonction de définition de MACRO d'édition
(DMP <at> <svar> <e1> ... <eN>)

DMP associe au symbole atomique **<at>** une MACRO d'édition. Cette MACRO est rangée sur la ***P-LIST*** du symbole **<at>** sous l'indicateur **PRETTY.**
La liaison des arguments de ce type de fonction est identique aux MACRO de l'évaluateur i.e. c'est l'appel tout entier de la MACRO d'édition qui est lié à la première variable du spécificateur de variables **<svar>.** Ces MACRO d'édition ont un mode de lancement qui diffère du lancement des MACRO de l'évaluateur : si durant une édition la fonction **PPRIN** rencontre une liste dont le CAR est un symbole atomique qui possède une définition de MACRO d'édition, alors cette liste n'est pas éditée normalement mais la MACRO d'édition est invoquée à la place. C'est cette MACRO qui réalisera l'édition. Il faut noter que cette MACRO d'édition peut se trouver à n'importe quelle profondeur de l'objet à éditer.

Si la fonction **PPRIN** n'éditait pas les appels de la fonction **(QUOTE <s>)** sous la forme **'<s>**, il serait très facile de le réaliser au moyen de la définition de la MACRO d'édition suivante :

```
(DMP QUOTE (l)
    (IF (CDDR l)
        (PRIN l)
        (PRINCH "'")
        (PPRIN (CADR l))))
```

De même pour réaliser l'édition de la fonction **(LIST e1 ... eN)** sous la forme **[e1 ... eN]**, il suffit de définir la MACRO d'édition suivante :

```
(DMP LIST (l)
    (PRINCH "[")
    (SETQ l (CDR l))
    (WHILE (LISTP l)
        (PPRIN (NEXTL l))
        (IF l (PRINCH " ")))
    (PRINCH "]"))
```

Ces MACRO d'édition vont également être utilisées pour restituer en sortie certains macro-caractères d'entrée.

Soit le MACRO-caractère d'entrée | qui possède la définition suivante :

```
(DMC "|" () ['BAR (READ)])
```

Sa restitution en cas d'édition par la fonction **PPRIN** est réalisée par la MACRO d'édition suivante :

```
(DMP BAR (l) (PRINCH "|") (PPRIN (CADR l)))
```

Ainsi

```
? '|x
          est transformé en :
          (QUOTE (BAR x))
          qui évalué livre :
          (BAR x)
          qui est imprimé :
= |x
```

6.7 LES COMPOSITIONS SPECIALES.

L'utilisation des 16 fonctions de sortie de notre modèle permet de réaliser n'importe quel type d'impression spéciale. Ces impressions peuvent être de type :
- algébriques (tels les impressions en GCOL, MLISP ou RLISP)
- documentaires (tel le CROSS-REFERENCE en VLISP 10 [CHAILLOUX 78c])
- symboliques (telle la notation RAINBOW [GREUSSAY 79a])

Abordons le problème de l'impression des objets circulaires ou partagés. Ces objets sont construits très facilement à partir de listes existantes en utilisant les fonctions de modifications physiques telles que **NCONC**, **RPLACA** ou **RPLACD**.
Le classique **(NCONC** <l> <l>) est le moyen le plus facile de fabriquer une liste circulaire en forçant dans le CDR du dernier doublet de la liste <l> un pointeur sur le début de cette même liste <l>.

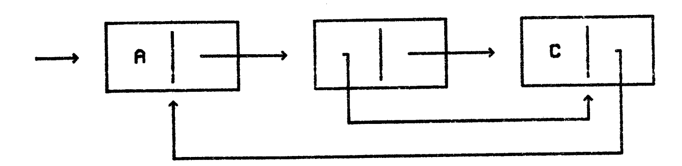
Etudions la structure de liste fabriquée par la séquence suivante :

```
(PROGN
   (SETQ L '(A () C))
   (RPLACA (CDR L) (CDDR L))
   (RPLACD (CDDR L) L)
   'L)
```

Il n'existe pas de représentation externe VLISP de cette structure. L'utilisation de la notation vue au chapitre 2 permet de donner une représentation de cette liste :

$$\{ \quad x{:}C\left|\begin{matrix}A\\ y\end{matrix}\right. \; , \quad y{:}C\left|\begin{matrix}z\\ z\end{matrix}\right. \; , \quad z{:}C\left|\begin{matrix}C\\ x\end{matrix}\right. \quad \}$$

Il existe une autre représentation des listes dans laquelle chaque doublet de liste est contenu dans une <u>double boîte</u> dont la partie gauche contient la composante CAR et la partie droite la composante CDR. Voici cette même structure représentée dans cette notation.



L'utilisation des fonctions de limitation d'impression pour autant qu'elles stoppent l'impression ne permettent pas de voir quels sont les doublets partagés.

avec PRINTLINE = 2

```
L = (A (C A (C A (C A (C A (C A (C A (C A (C A (C A (C A
       (C A (C A (C A (C A (C A (C A (C A (C A (C A (C A (C
```

avec PRINLENGTH = 6

```
L = (A (C A (C A (C ...) ...) ...) ...)
```

avec PRINLEVEL = 2

```
L = (A (C A & C A & C A & C A & C A & C A & C A & C A & C A & C A
```

Il faut donc disposer de fonctions spécialisées pour imprimer les listes circulaires ou partagées. Voici le texte de la fonction d'édition **CPRIN** et de la fonction d'impression **CPRINT** d'une telle liste. A l'occurrence d'un doublet partagé ou d'une liste circulaire, l'édition s'arrête et des points de suspension sont imprimés suivis d'une parenthèse fermante, ce qui permet de continuer l'impression.

```
; Fonctions d'impressions circulaires simples ;

(DE CPRINT (l)
   ; Fonction d'impression de l ;
   (CPRIN l)
   (TERPRI)
   l)

(DE CPRIN (l)
   ; Fonction d'édition de l ;
   (CPRIN1 l ()))

(DE CPRIN1 (l vus)
   ; Fonction auxillaire ;
   ; vus contient la liste des objets déjà visités ;
   (LET (l l)
      (IF (ATOM l)
         ; Les atomes peuvent être partagés ;
         (PRIN l)
         ; En cas de liste ;
         (PRINCH "(")
         (WHILE (LISTP l)
            (IF (MEMQ l vus)
               ; C'est un objet déjà visité ;
               (EXIT (PRIN "...)"))
               ; C'est un objet nouveau ;
               (NEWL vus l)
               ; Récurse sur les CAR ;
               (SELF (NEXTL l))
               (IF l (PRINCH " "))))
         (IF (NULL l)
            ; La liste se termine bien ;
            ()
            ; C'est une paire pointée ;
            (PRINCH ".")
            (PRINCH " ")
            (PRIN l)))
         (PRINCH ")")))))
```

En utilisant cette fonction, la liste L fabriquée au début du paragraphe peut s'imprimer :

```
(CPRINT L)  ☞  (A (C ...) ...)
```

Ces fonctions permettent donc d'imprimer n'importe quelle structure de liste. Une amélioration notable consiste à référencer (au moyen de numéro entre accolades) les doublets partagés.

Voici les fonctions **CCPRIN et CCPRINT** réalisant de telles impressions.

```
; Fonctions d'impressions circulaires avec références ;

(DE CCPRINT (l)
   ; Imprime l'objet l ;
   (CCPRIN l) (TERPRI) l)

(DE CCPRIN (l)
   ; Edite l'objet l ;
   (CCPRIN1 l () ()))

(DE CCPRIN1 (l vus vusplusieurs)
   ; 1ère passe qui recherche les objets partagés. ;
   ; Ils sont rangés dans : vusplusieurs  ;
   ; vus : contient la liste de tous les objets visités ;
   (LET (l l)
      (IF (ATOM l)
          ()
          (WHILE (LISTP l)
             (IF (MEMQ l vus)
                 (EXIT
                    (IF (NOT (MEMQ l vusplusieurs))
                        (NEWL vusplusieurs l))))
             (NEWL vus l)
             (SELF (NEXTL l)))))
   (SETQ vus ())
   ; 2ème passe qui édite à proprement parlé en utilisant ;
   ; la liste vusplusieurs fabriquée  durant la 1ère passe ;
   (LET (l l)
      ; l doit être locale ;
      (IF (ATOM l)
          (PRIN l)
          (PRINCH "(")
          (WHILE (LISTP l)
             (IF (MEMQ l vusplusieurs)
                 (PROGN
                    (PRINCH "{")
                    ; le no de l'objet partagé est l'inverse ;
                    ; de sa position dans vusplusieurs ;
                    (PRIN (LENGTH (MEMQ l vusplusieurs)))
                    (PRINCH "}")))
             (IF (MEMQ l vus)
                 ; C'est un objet déjà visité ;
                 (EXIT (PRIN "...)"))
                 ; C'est un objet nouveau ;
                 (NEWL vus l)
                 (SELF (NEXTL l))
```

```
            (IF L (PRINCH " "))))
    (IF L
      (PROGN
         (PRINCH ".")
         (PRINCH " ")
         (PRIN L)))
    (PRINCH ")")))))
```

Cette fonction imprime une liste circulaire de la manière suivante :

(SETQ X '(A B C)) ☞ (A B C)

(CCPRINT (NCONC X X)) ☞ ({1}A B C {1}...)

En utilisant cette même fonction d'impression, la liste L fabriquée au début du paragraphe peut s'imprimer :

(CCPRINT L) ☞ ({1}A ({2}C {1}...) {2}...)

CHAPITRE 7
LES CONCEPTS AVANCES DE L'INTERPRETATION

Ce chapitre va compléter la description de notre évaluateur donné au chapitre 4. Nous y verrons un nouvel interprète, plus complexe, qui va permettre :
- d'évaluer itérativement certaines classes de récursions terminales
- d'implémenter les fonctions de contrôle dynamique **EXIT** et **SELF**
- d'implémenter les fonctions de définition dynamique **WHERE** et **ESCAPE**
- de traiter les variables fonctions
- d'utiliser le concept de CHRONOLOGIE.

Le texte complet de cet évaluateur est donné à l'appendice F.

Cet évaluateur va mettre en jeu une structure de contrôle plus puissante constituée d'une pile polymorphique.

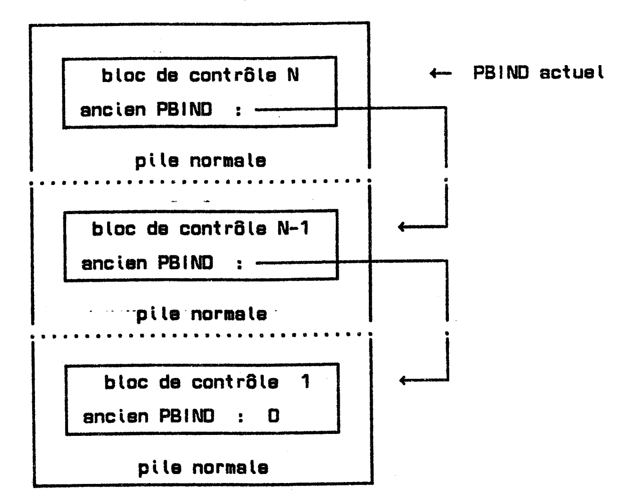
7.1 LA STRUCTURE DE LA PILE.

La pile a été utilisée jusqu'à maintenant d'une manière très simple : les opérations d'accès de la pile étaient obtenues au moyen des instructions machine VCMC2 : **PUSH**, **POP**, **TOPST** et **XTOPST**. Seul le dernier élément de la pile était consultable. L'implémentation des structures de contrôle dynamique va induire l'utilisation d'une pile polymorphique dans laquelle il va être possible de consulter outre le sommet de pile, des zones plus profondes de la pile. Cette consultation ne doit bien évidemment pas être destructive.

7.1.1 Structure de la pile polymorphique

Cette pile polymorphique consiste à construire dans la pile elle-même des blocs de contrôle. Ces blocs de contrôle sont des zones de la pile qui ne sont pas gérées comme une pile et qui sont liées entre elles de la manière suivante :
- l'adresse du dernier bloc de contrôle construit se trouve dans la variable **PBIND**
- chaque bloc de contrôle contient l'adresse du bloc de contrôle précédent
- l'adresse du bloc précédent le premier bloc de contrôle construit est 0.

Voici le schéma de cette pile polymorphique :



Ces blocs de contrôle vont avoir une durée de vie comparable aux éléments de la pile : les blocs seront construits (empilés) et détruits (dépilés) en observant une technique de LIFO stricte. C'est ce qui différencie notre structure de contrôle des piles ***spaghetti*** [BOBROW 73b] voire ***macaroni*** [STEELE 77] dans lesquelles l'allocation de la pile s'effectuait par blocs dynamiques et necessitait du même coup l'emploi d'un récupérateur spécial lourd et encombrant.

7.1.2 Structure d'un bloc de contrôle

Un bloc de contrôle est un ensemble de données structurées qui est construit dans les occasions suivantes :

- à l'entrée d'une fonction de type EXPR/FEXPR/MACRO
- à l'entrée d'une fonction **WHERE** ou **ESCAPE**
- à l'entrée d'une fonction **LETF**
- à la création d'une nouvelle CHRONOLOGIE

Dans tous ces cas, les données contenues dans ces blocs de contrôle n'auront pas la même structure; on est donc amené à typer ces blocs au moyen d'un numéro de type de bloc.

Ce numéro de type de bloc doit être très facilement accessible et c'est pourquoi il occupe toujours un emplacement fixe à l'intérieur de chaque bloc de contrôle : dans notre implémentation ce numéro se trouve toujours en première position dans le bloc i.e. pointé directement par **PBIND**.

Cette position privilégiée permet également d'accélérer le module de destruction des blocs de contrôle (le module **UNBIND:**). Ce module doit positionner le pointeur de pile en tête du bloc de contrôle puis réaliser un aiguillage vers un sous-module dépendant du type du bloc. Ceci est aisément réalisé au moyen des 2 instructions VCMC2 suivantes :

```
UNBIND:   SSTACK   (@ PBIND)
          JUMPX    (TUNBIND),TST

TUNBIND:  DATA     adresse si bloc de type 0
          DATA     adresse si bloc de type 1
          DATA     adresse si bloc de type 2
          ....           ....
```

La première instruction positionne le pointeur de pile en tête du dernier bloc de contrôle construit et la seconde réalise l'aiguillage dans la table des étiquettes **TUNBIND** en fonction du type de bloc dépilé.

Chaque sous-module doit repositionner la variable **PBIND** sur le bloc de contrôle précédent.

7.1.3 Réalisation des blocs de contrôle

Dans notre modèle d'implémentation, les blocs de contrôle se construisent avec des instructions d'empilement **PUSH** ou **XTOPST**. Toutefois la nécessité de construire facilement des blocs de contrôle dans la pile se fait de plus en plus sentir et aujourd'hui des machines apparaissent avec de nouvelles instructions spécialisées dans la manipulation de ces blocs de contrôle. Par exemple le nouveau micro-processeur de MOTOROLA le MC68000 [MOTOROLA 79], disponible en France à la fin de l'année 1979, possédera le couple d'instructions spécialisées **LINK** et **UNLNK**. Ces instruction permettront de réaliser directement les opérations de fabrication d'un bloc de contrôle dans la pile.

Voici la description de ces instructions pour lesquelles **reg** est un registre quelconque (l'équivalent du pointeur **PBIND** dans la machine VCMC2) et **n** un nombre entier :

LINK reg,n	réalise	PUSH reg SP → reg SP + n → SP
UNLNK reg	réalise	SSTACK reg POP reg

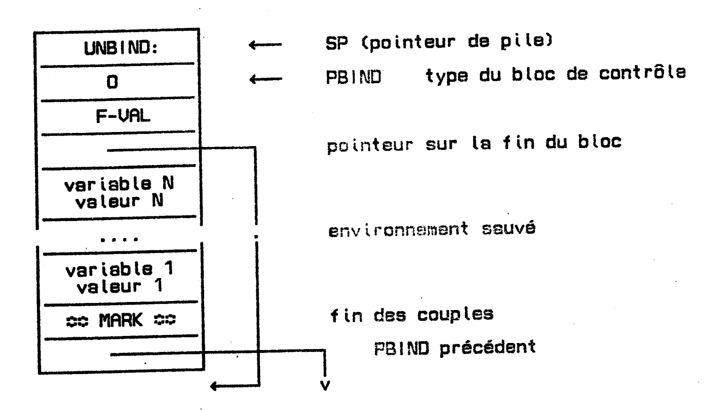
A moindre titre les instructions **CALLG** et **CALLS** du VAX11/780 [DEC 78e] permettent des appels de procédures complexes mettant en jeu :
- le pointeur de pile lui-même
- le registre contenant le lien vers le bloc de contrôle précédent (le Frame Pointer)
- et un registre pointant sur le bloc argument (qui est une partie du bloc de contrôle).

Ces instructions pourront bien évidemment être utilisées à la construction des blocs de contrôle.

7.2 LE BLOC DE CONTROLE DE EVAL

Nous avons vu au chapitre 4 que la fonction interprète **EVAL** devait sauver à chaque appel de fonction de type EXPR/FEXPR/MACRO la partie de l'environnement (i.e. les couples variable-valeur) qui était modifiée par l'appel de la fonction. Cet environnement était sauvé dans la pile. Dans ce nouvel interprète, l'environnement va être sauvé dans un bloc de contrôle.

Voici la structure du bloc de contrôle fabriqué par EVAL à l'entrée d'une fonction de type EXPR/FEXPR/MACRO :



Ce bloc de contrôle contient, en plus de l'environnement sauvé :
- le type du bloc de contrôle (qui est ici le type **0**). pointé par **PBIND**
- la ***F-VAL*** de la fonction à exécuter
- un pointeur sur la fin du bloc de contrôle (ce pointeur est utilisé pour accéder directement au dernier mot empilé AVANT la fabrication du bloc de contrôle)
- un pointeur sur le bloc de contrôle précédent.

Toutes ces informations supplémentaires vont être utilisées pour implémenter les possibilitées nouvelles de l'évaluateur :
- interprétation itérative des récursions terminales,
- implémentation des fonctions **EXIT** et **SELF**.

7.3 DIFFERENTS TRAITEMENTS DE LA RECURSIVITE.

Dans certains cas de récursions terminales de fonctions de type EXPR, l'interprète VLISP effectue dynamiquement un traitement particulier qui lui permet d'interpréter itérativement les appels récursifs de type *{Note 1}* :

- post-récursions normales (ou NPR)
- post-récursions mutuelles (ou CPR)

Les post-récursions enveloppées ne sont pas traitées dans cette étude.

L'interprétation itérative d'appels NPR ou CPR va s'effectuer sans consommation de ressources de type pile ou doublets de liste.
Dans la fonction suivante (qui cherche le dernier élément d'une liste) l'appel récursif de la dernière ligne est un appel récursif terminal.

```
(DE FOO (L)
    (IF (NULL (CDR L))
        (CAR L)
        (FOO (CDR L))))
```

Les appels de la fonction **FOO** consommeront le même espace pile et n'utiliseront aucun doublet de liste, quelque soit la taille de la liste **L**.

La puissance de cette méthode réside dans la recherche des récursions terminales qui est réalisée dynamiquement dans l'interprète par une consultation extrêmement rapide de l'état de la pile.

L'interprète doit pouvoir effectuer deux tests à l'évaluation d'une fonction de type EXPR :

- le test de position terminale
- le test de récursion

{Note 1} cette typologie a été introduite par P. GREUSSAY [GREUSSAY 77] et une approche formelle de l'élimination de ces fausses récursivités est donnée dans [DURIEUX 79].

7.3.1 Le test de position terminale

L'évaluation est en position terminale si on trouve dans le sommet de pile l'adresse du module qui délie les blocs de contrôle (i.e. l'adresse **UNBIND:**). En effet le bloc de contrôle doit être défait au retour de la dernière évaluation. Ce test est donc très simple, il faut toutefois prendre garde au niveau des fonctions appelant l'interprète de ne pas masquer cet état terminal : la dernière évaluation d'une fonction (l'évaluation de la valeur de la fonction) doit être appelée par une continuation

[JUMP (EVAL)] ou **[JUMP (EVCAR)]**) *{Note 1}*.

Cette précaution doit s'appliquer à tous les séquenceurs i.e. aux fonctions **PROGN**, **AND** et **OR**.

Voici le séquenceur principal **PROGN** qui appelle la dernière évaluation au moyen de la continuation **[JUMP (EVCAR)]**) :

```
{1}  PROGN:  CDR    A1,A2
{2}          FLIST  A2,,[JUMP (EVCAR)])
{3}          PUSH   A2,,[CALL (EVCAR)])
{4}          POP    A1,,[JUMP (PROGN)])
```

Qui appelle bien la dernière évaluation au moyen d'un **[JUMP (EVCAR)]** après avoir libéré la pile.

L'écriture suivante de la fonction **OR** est correcte mais ne permet plus de tester la position terminale d'une évaluation car au moment de la dernière évaluation la pile n'est pas vide mais contient le CDR de la liste des arguments (i.e. NIL) :

{Note 1} cette continuation est un branchement vers l'instruction :

EVCAR: CAR A1,A1,[JUMP (EVAL)]

```
OR1:  POP    A1
OR:   FLIST  A1,,[RETURN]
      CDR    A1,TST,[CALL (EVCAR)]
      TNIL   A1,,[JUMP (OR1)]
      POP    A2,,[RETURN]
```

Pour permettre le test de position terminale il faut écrire ce module de la manière suivante :

```
OR:      CDR    A1,A2
         FLIST  A2,,[JUMP (EVCAR)]
         PUSH   A2,,[CALL (EVCAR)])
         FNIL   A1,,[JUMP (OR1)])
         POP    A1,,[JUMP (OR)])
OR1:     POP    A2,,[RETURN]
```

7.3.2 <u>Le test de récursion</u>

Une fois le test de position terminale effectué, il faut réaliser le test de récursion. Cette récursion peut être :
- une récursion normale (une fonction s'appelle elle-même)
- une récursion mutuelle (une fonction appelle une autre fonction en position terminale qui rappelle une autre fonction en position terminale qui .... et qui appelle la première fonction en position terminale).

Il s'agit donc d'une suite d'appels répondant au schéma :

```
(DE FOO1 (...)   ... (FOO2 ...))
(DE FOO2 (...)   ... (FOO3 ...))
        ...          ...
(DE FOON-1 (...) ... (FOON ...))
(DE FOON (...)    ... (FOO1 ...))
```

Voici un exemple d'appels récursifs mutuels tiré de [GREUSSAY 77] pp. 179. **PERMS** est une fonction qui imprime la suite des permutations de l'intervalle [1,n] en ordre lexicographique.

```
(DE PERMS (n)
   (F 0 NIL))

(DE F (niv e)
   (IF (< niv n)
      (H (+ niv 1) (CONS 1 e))
      (PRINT (REVERSE e))
      (G niv (CAR e) (CDR e))))

(DE G (niv x e)
   (IF (= x n)
      (IF e
         (G (- niv 1) (CAR e) (CDR e))
         'FINI)
      (H niv (CONS (+ x 1) e))))

(DE H (niv e)
   (IF (MEMQ (CAR e) (CDR e))
      (G niv (CAR e) (CDR e))
      (F niv e)))

  ? (PERMS 3)
    (1 2 3)
    (1 3 2)
    (2 1 3)
    (2 3 1)
    (3 1 2)
    (3 2 1)
  = FINI
```

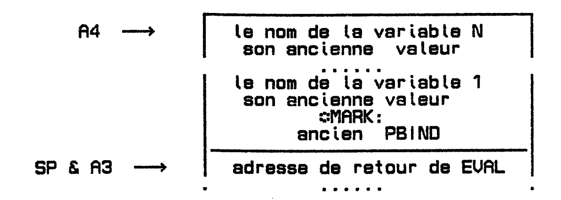
Tous les appels croisés des fonctions **F**, **G** et **H** sont de type récursifs mutuels donc l'évaluation d'un appel de la fonction **PERMS** nécésitera le même espace pile quelque soit la valeur du nombre **n**.

Le test de récursion simple est réalisé par comparaison de la ***F-VAL*** de la fonction à évaluer avec la ***F-VAL*** de la dernière fonction évaluée et dont la ***F-VAL*** se trouve dans le bloc de contrôle précédent.
Le test de récursion mutuelle est réalisé en répétant ce test pour tous les blocs de contrôle qui se trouvent en position terminale.

7.3.3 Réalisation de l'interprétation itérative

La réalisation de l'interprétation itérative des récursions terminales et mutuelles est effectuée dans notre modèle avec un minimum d'instructions supplémentaires : les tests de position terminale et de récursion ne s'opèrent qu'APRES avoir construit dans la pile le bloc de contrôle et réalisé les liaisons (i.e. avant de lançer l'évaluation du corps de la fonction telle qu'elle est décrite au chapitre 4).

Voici le contenu de la pile et la position des différents pointeurs APRES la liaison des arguments. Le contenu de la pile est identique à celui obtenu au paragraphe 4.6.4 seul le pointeur **PBIND** à été préalablement empilé.



Le registre A1 contient la ***F-VAL*** de la fonction. La réalisation de l'interprétation itérative est réalisée par NON-REMISE A JOUR du pointeur de pile qui va rester positionné sur l'adresse de retour de EVAL. L'ébauche de bloc de contrôle construite dans la pile va donc être perdue (et la place occupée rendue à la pile).

Voici le code de l'implémentation de l'interprétation itérative :

```
; Test de récursion terminale (normale ou mutuelle)
; des fonctions de type EXPR

EVEXP6:
    NEQ    TST,(UNBIND),[JUMP (EVEXPN)]   ;  pas terminal
    JUMPX (TEVEX),TST   ; aiguillage sur le type de bloc

TEVEX:
    DATA (TEVEXL)      ; type 0 : bloc EXPR/FEXPR/MACRO
    DATA (EVEXPN)      ; type 1
    .... .........     . .... .

TEVEXL:                       ; bloc EXPR/FEXPR/MACRO
    EQ     A1,TST,[JUMP (EVEXP9)] ; c'est récursif !
    SSTACK TST [JUMP (EVEXP6)]    ; passe à la fin du
                             ; bloc et test le bloc précédent

EVEXP9:                           ; c'est itératif
    SSTACK A3                     ; ajuste la pile
    CDR    A1,A1,[JUMP (PROGN)]   ; et réalise un JUMP!

EVEXPN:                  ; appel normal
    SSTACK A4            ; repasse en début de bloc
    PUSH   A3            ; empile l'adresse de fin du bloc
    PUSH   A1            ; empile la F-VAL
    PUSH   '0            ; empile le code du bloc
    STACK  (@ PBIND)     ; sauve le pointeur de pile
    CDR    A1,A1,[JUMP (EXEC)]    ; évalue le corps
```

Les performances de notre implémentation sont excellentes car les tests demandent :
- 1 instruction si l'appel n'est pas terminal
- 3 instructions pour déterminer si un appel est une récursion terminale
- 3*n instructions pour déterminer si un appel est une récursion mutuelle (n étant le nombre d'appels de fonctions terminales différentes de la fonction de l'appel).

Ces tests ralentissent donc l'évaluateur de manière insignifiante.

7.4 LES FONCTIONS DE CONTROLE DYNAMIQUES.

VLISP autorise deux nouvelles fonctions de contrôle dynamique, compatibles avec la structure de la pile :

- la contruction **EXIT** *{Note 1}* qui permet de sortir d'une fonction de type EXPR/FEXPR/MACRO.

- la construction **SELF** *{Note 2}* qui permet de rappeler la dernière fonction invoquée de type EXPR/FEXPR/MACRO.

Ces deux nouvelles fonctions vont être aisément implémentées du fait de la structure du bloc de contrôle fabriqué par EVAL à l'entrée des fonctions de type EXPR/FEXPR/MACRO.

7.4.1 Implémentation de la fonction EXIT

La fonction EXIT permet de retourner de la dernière fonction appelée. C'est donc l'équivalent de la fonction **RETURN** des anciennes formes **PROG.** Ainsi la fonction **FIND1NUMBER** retourne en valeur le premier nombre d'une liste, ou bien **0** si la liste n'en contient pas.

```
(DE FIND1NUMBER (l)
    (WHILE (LISTP l)
        (IF (NUMBP (CAR l))
            (EXIT (CAR l))
            (SETQ l (CDR l))))
    0)
```

L'utilisation de la fonction **EXIT** permet de retourner une valeur en abandonnant tout ce qui était en cours, ici la fonction **WHILE.** L'implémentation de cette fonction est donc évidente : il suffit de vider la pile de tout ce qui se trouve au dessus du dernier bloc de contrôle construit puis dans ce nouvel état de la pile, d'évaluer la

---

{Note 1} cette construction est apparue pour la première fois en VLISP-10 [CHAILLOUX 78c] sous le nom de **LESCAPE.**

{Note 2} cette construction a été introduite par P. GREUSSAY [GREUSSAY 77]

valeur qui doit être retournée par la fonction. Il faut toutefois tester s'il existe bien un bloc de contrôle dans la pile (i.e. si au moins une fonction de type EXPR/FEXPR/MACRO) a été appelée.

```
EXIT:   EQ       '0,(@ PBIND),[JUMP (EXITER)])
        SSTACK   (@ PBIND)
        PUSH     (UNBIND),,[JUMP (PROGN)])
```

La première instruction teste l'existence d'un bloc de contrôle, la seconde positionne le pointeur de pile sur le type du dernier bloc de contrôle contruit et la troisième prépare le retour de la fonction et évalue la valeur à retourner.

Cette implémentation permet en outre d'utiliser la fonction **EXIT** pour forcer une évaluation itérative d'un appel récursif.

Ainsi dans la fonction **FIND2NUMBER** qui retourne le 1er nombre trouvée dans la plus profonde sous-liste d'une liste donnée :

```
(DE FIND2NUMBER (l)
   (WHILE (LISTP l)
      (COND
        ((LISTP (CAR l)) (EXIT (FIND2NUMBER (CAR l))))
        ((NUMBP (CAR l)) (EXIT (CAR l))))
      (SETQ l (CDR l))))
```

L'appel **(EXIT (FIND2NUMBER (CAR l)))** est, outre un retour de la fonction, un forçage d'interprétation itérative de **(FIND2NUMBER (CAR l))**.
En effet le pointeur de pile est mis au niveau du dernier bloc de contrôle construit dans la pile <u>AVANT</u> évaluation de la valeur retournée, qui est donc <u>TOUJOURS</u> évaluée en position terminale.

7.4.2 Implémentation de la fonction SELF

Cette fonction permet de rappeler la dernière fonction invoquée de type EXPR. Nous réaliserons l'implémentation de cette fonction par recherche de la ***F-VAL*** de la fonction dans le dernier bloc de contrôle construit par **EVAL** puis par l'évaluation de cette fonction avec les arguments transmis à **SELF**.

Voici le code de l'implémentation de la fonction **SELF**. Le traitement de l'erreur **SELFER** n'est pas donné.

```
SELF:
    STACK   A2                     ; sauve le pointeur courant
    MOVE    (@ PBIND),A4           ; récup début de bloc
SELF1:
    EQ      A4,'0,[JUMP (SELFER)] ; erreur fatale!
    SSTACK  A4                     ; pointe sur nouveau bloc
    JUMPX   (TSELF),TST            ; aiguillage sur type de bloc

TSELF:
    DATA @  (SELFL)                ; Type 0 : bloc EXPR
    ....    .......                ; Type 1 :

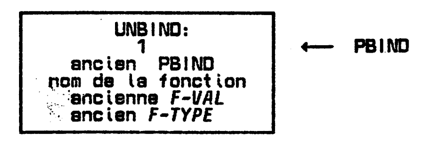
SELFL:
    POP      A4                    ; récup la Fval empilée
    SSTACK   A2                    ; revient au début du SELF
    PUSH     A4,,[JUMP (EVEXP)]]   ; empile la nouvelle FVAL
```

7.5 LES FONCTIONS DYNAMIQUES.

Les fonctions définies dynamiquement sont une particularité majeure de notre modèle. Il est en effet possible de réaliser des définitions temporaires de n'importe quel type de fonction. Ces définitions temporaires sont obtenues par liaison dynamique des propriétés naturelles *F-VAL* et *F-TYPE* des symboles atomiques associés aux fonctions.

Cette liaison dynamique est réalisée d'une manière identique à la liaison dynamique des variables : il y a fabrication d'un bloc de contrôle dans la pile (ici le type du bloc de contrôle est 1) qui contient les anciennes valeurs des propriétés naturelles permettant ainsi de rendre fluides les fonctions elles-même. Un nouveau type de bloc de contrôle est nécessaire car la construction et la destruction de ce type de bloc est différente de celles des blocs de contrôle fabriqués par **EVAL**.

Voici la structure de ce nouveau type de bloc de contrôle :



Et voici le module de destruction d'un tel bloc :

```
UNBIND1:
   POP     (@ PBIND)        ; repositionne PBIND
   POP     A4               ; enlève le nom
   SFVAL   TST,A4           ; restaure l'ancienne F-VAL
   SFTYP   TST,A4,[RETURN]  ; restaure l'ancien F-TYP
```

Deux fonctions VLISP vont construire de tels blocs :
- la fonction de définition dynamique **WHERE**
- la fonction d'échappement **ESCAPE**.

7.5.1 Implémentation de la fonction **WHERE**

WHERE permet de redéfinir dynamiquement (i.e. d'associer dynamiquement à un symbole atomique) une fonction de n'importe quel type. Cette définition sera active le temps de l'évaluation d'une ou plusieurs expressions et à la fin de l'évaluation l'ancienne fonction associée au symbole sera restaurée.

Il existe deux formes d'appel de la fonction **WHERE** :

```
(WHERE (<at> <FTYPE> <FVAL>)  <e1> ... <eN>)
    (WHERE (<at> <FVAL>)  <e1> ... <eN>)
```

Les deux vont associer au symbole atomique **<at>** une définition de fonction. La valeur de définition de la fonction est **<FVAL>** qui est <u>évalué</u> et le type de la fonction est <FTYPE> dans la première forme et EXPR implicitement dans la deuxième forme (cette forme abrégée a été ajoutée dans le système car étant la plus utilisée).
Cette liaison réalisée **WHERE** va évaluer les expressions **<e1>** ... **<eN>** et retourner en valeur la valeur de la dernière évaluation. Toutefois avant de retourner à l'appelant, l'ancienne fonction associée à l'atome va être restaurée.

Voici la réalisation de la fonction **WHERE** dans notre modèle :

```
WHERE:  CAR    A1,A2              ; A2 ← la nouvelle définition
        CDR    A1,TST             ; sauve le corps à exécuter
        CAR    A2,TST             ; sauve le nom de la fonction
        CDR    A2,A2              ; A2 ← nouvelle fonction
        CAR    A2,A1              ; A1 ← le nouveau F-type
        TNUMB  A1,,[JUMP (WHERE1)]   ; il y a un FTYP explicite
        NOP    ,,[CALL (EVAL)] ; évalue la F-val
        MOVE   '7,A4,[JUMP (WHERE2)] ; par défaut FTYP=7 : EXPR
WHERE1: PUSH   A1                 ; sauve le F-type
        CDR    A2,A1,[CALL (EVCAR)]) ; évalue la F-val
        POP    A4                 ; A4 ← le F-type
WHERE2: MOVE   A1,A3              ; A3 ← la F-val
        POP    A2                 ; A2 ← le nom
        POP    A1                 ; A1 ← le corps
WHERE3:                           ; Liaison dynamique avec :
                                  ;     A1 ← le corps à exécuter
                                  ;     A2 ← le nom de la fonction
                                  ;     A3 ← la nouvelle F-VAL
                                  ;     A4 ← le nouvel F-TYP
        FTYP   A2,TST             ; sauve l'ancien F-TYP
        FVAL   A2,TST             ; sauve l'ancienne F-VAL
```

```
        PUSH   A2              ; sauve le nom de la fonction
        SFTYP  A4,A2           ; force le nouveau type
        SFVAL  A3,A2           ; force la nouvelle F-VAL
        PUSH   (@ PBIND)       ; sauve l'adresse du bloc preced.
        PUSH   '1              ; type du bloc
        STACK  (@ PBIND)       ; actualise le nouveau PBIND
        PUSH   (UNBIND),,[JUMP (PROGN)]
                 ; prépare le retour et exécute le corps
```

7.5.2 Implémentation de la fonction **ESCAPE**

L'implémentation de la fonction **ESCAPE** *{Note 1}* s'apparente dans notre modèle à celle du **WHERE**. En effet il s'agit là encore d'une définition dynamique. Toutefois le nouveau *F-TYPE* de la fonction est un F-TYPE spécial, (dont la valeur est 10) qui indique que la fonction est une fonction d'échappement.

```
ESCAPE: MOVE   '10,A4     ; F-TYP de type ESCAPE
        CAR    A1,A2      ; nom de la fonction
        CDR    A1,A1      ; corps à exécuter
        MOVE   A2,A3,[JUMP (WHERE3)] ; F-VAL = le nom
```

L'évaluation des fonctions d'échappement est réalisée par **EVAL**. Voici le texte du module spécialisé qui traite ce type de fonction dans lequel **UNBINP** est le nom du module qui détruit le dernier bloc de contrôle construit dans la pile (ce module se différencie du module **UNBIND** car il suppose que **A3** contient l'adresse de retour du module).

{Note 1} on trouvera une description de cette fonction dans [GREUSSAY 77] pp. 148-151.

```
EVESC:                                ;Evaluation d'un ESCAPE
     PUSH    A2,,[CALL (PROGN)]       ; sauve le nom du ESCAPE
     POP     A2                       ; récupère le nom
EVESC1:                               ; délie un bloc
     EQ      '0,(@ PBIND),[JUMP (ERESC)] ; il n'y en a plus !?!
     SSTACK (@ PBIND)                 ; pile en début de bloc
     TOPST   A3                       ; récupère le sommet de pile
     EQ      A3,'1,[JUMP (EVESC3)]    ; c'est un bloc WHERE/ESCAPE
     MOVE    (EVESC1),A3,[JUMP (UNBINP)] ; delie ce bloc simplement
EVESC3:
     MOVE    (EVESC4),A3,[JUMP (UNBINP)] ; délie ce bloc
EVESC4:
     NEQ     A2,A4,[JUMP (EVESC1)]    ; ce n'est pas la bonne fonction
     NOP     ,,[RETURN]               ; c'est tout bon
```

L'utilisation des blocs de contrôle permet donc de gérer avec une grande facilité la restauration de l'environnement. Cette restauration va demander la destruction d'un nombre quelconque de blocs de contrôle.

7.6 LES VARIABLES FONCTIONS.

Il existe dans l'interprète VLISP des variables internes qui contiennent des valeurs susceptibles d'être consultées ou modifiées par l'utilisateur. Par exemple, la base de conversion des nombres en entrée utilisée dans l'interprète doit pouvoir être lue et/ou modifiée, de même que la longueur d'une ligne en sortie.

Ces variables internes, peu nombreuses (notre système n'en possède qu'une quinzaine) contrôlent principalement les mécanismes d'entrée/sortie. L'accès à ces variables internes augmente considérablement la puissance et la généralité du système.

Lorsqu'il s'agit de donner accès, tant en consultation qu'en modification, à ces variables internes, deux stratégies sont disponibles :

1) Faire de ces variables internes des variables VLISP. Dans ce cas le système suppose que la valeur de ces variables internes se trouve dans la *C-VAL* des variables VLISP correspondantes.
   Côté utilisateur, la consultation et la modification de ces variables est très aisée, il suffit d'accéder à leur *C-VAL*. Toutes les propriétés des variables sont conservées et en particulier leur fluidité.

   Appelons OBASE, la variable interne contenant la valeur de la base de numération des nombres en sortie. Et voici pour illustrer cette stratégie une fonction qui imprime son 1er argument <s> dans la base de numération **OBASE**, fournie en 2ème argument.

Définition :

```
(DE PRINT-BASE-X (<s> OBASE)
   (PRINT <s>))
```

Utilisation :

(SETQ n 1000)	☞	1000
(PRINT-BASE-X n 8)	☞	1750
(PRINT-BASE-X n 11)	☞	82A
(PRINT-BASE-X n 16)	☞	3E8
OBASE	☞	10

Du fait de la fluidité de la variable OBASE, au sortir de cette fonction d'impression, l'ancienne valeur de OBASE est restaurée.

En revanche côté système, l'accès est très ralenti car il faut utiliser la *C-VAL* de l'atome et cette *C-VAL* contient toujours un pointeur sur un objet VLISP qu'il faut convertir en objet manipulable par la machine (ce qui est particulièrement lent dans le cas des atomes numériques). Toutefois le gros défaut de cette stratégie est d'obliger le système à contrôler la validité de la valeur de la variable. En effet l'appel **(SETQ OBASE 0)** en plus de provoquer une erreur de la machine pour tentative de division par 0 (car les conversions en sortie se font par divisions successives), rendra le système incapable par la suite d'imprimer de nouveaux nombres. L'effet est parfois encore plus désastreux comme dans le cas où les impressions doivent se réaliser dans un tampon de longueur nulle voire négative, ce qui empêche toute impression y compris celle du message d'erreur.

Ce problème de contrôle de validité des variables internes se pose donc de façon très aigue et le système se doit de vérifier la validité des valeurs actuelles avant tout accès aussi bien en lecture qu'en écriture car les modifications peuvent intervenir à n'importe quel moment.

2) Une deuxième stratégie consiste à associer à ces variables internes des fonctions d'accès particulières qui vont manipuler les variables internes. Ces fonctions d'accès permettent la consultation et la modification d'une variable interne suivant un processus commun : la consultation se réalise par appel de la fonction sans argument, la modification par appel de la fonction avec 1 argument évalué qui devient la nouvelle valeur de la variable interne.

```
(fonction-d'accès)           :   lecture
(fonction-d'accès  valeur)   :   écriture
```

En cas de modification des variables, ces fonctions testent la validité de l'argument ce qui évite de placer le système dans un état critique. Ces tests de validité ne se font qu'à chaque demande de modification explicite (i.e. qu'à chaque écriture).

Aucune de ces deux stratégies n'est idéale. La première ralentit trop le système et la seconde ne permet plus d'avoir des variables internes fluides.

Pour approcher les possibilités de la stratégie utilisant des variables, il est possible de macro-générer à chaque appel de type modification d'une fonction de variable interne (appelons la **<varint>**) par la valeur **<e>**, une séquence d'appels simulant la fluidité des variables :

1) sauvetage de l'ancienne valeur de la variable interne <varint> dans la variable VLISP <varint>.

   (SETQ <varint> (<varint>))

2) modification de la variable interne par <e>.

   (<varint> <e>)

   ...

3) exécution d'un programme avec la valeur de la variable interne modifiée.

   ...

4) restauration de l'ancienne valeur de la variable interne.

   (<varint> <varint>)

Voici la MACRO VLISP réalisant cet enchainement :

```
(DM VARINT (L)
   (RPLACB L
      ['LET
         [(CAADR L) [(CAADR L)]]
         [(CAADR L) (CADADR L)]
         ['PROG1
            ['PROGN . (CDDR L)]
            [(CAADR L) (CAADR L)]]]))
```

Redéfinissons la même fonction que précédemment :

```
(DE PRINTBASEX (s n)
   (VARINT (OBASE n)  (PRINT s))))))
```

Appel de la fonction :

(PRINTBASEX 1000 16) ☞ 3E8

Modification de la fonction :

(FVAL 'PRINTBASEX) ☞

```
((s n)
   (LET (OBASE (OBASE))
      (OBASE n)
      (PROG1 (PROGN (PRINT s))
         (OBASE OBASE))))
```

Cette macro-génération, qui montre bien les limites de la stratégie de fonctionnalisation de l'accès, laisse à désirer : elle est lente mais surtout n'a pas la puissance de la liaison des variables fluides. L'utilisation de la variable de même nom que la fonction ne permet plus les modifications imbriquées et les anciennes valeurs de la variable interne ne sont pas restaurées si des fonctions de contrôle dynamiques telles des fonctions **ESCAPE** sont invoquées.

C'est pourquoi nous avons donné à notre modèle un nouveau type de fonctions, les variables fonctions, qui permettent dans une stratégie de fonctionnalisation des variables internes, d'avoir la souplesse de la liaison des variables fluides.

Une variable fonction est une fonction d'accès à une variable interne. Elle est une fonction à part entière et peut par conséquent être redéfinie.

Pour réaliser la fluidité de cette variable fonction une nouvelle fonction de type FSUBR à été crée, la fonction de liaison des variables fonctions, de nom **LETF** :

```
(LETF (<varfnt> <e>) <e1> ... <eN>)
```

dans laquelle **<varfnt>** est le nom d'une variable fonction, **<e>** sa

nouvelle valeur et <e1> ... <eN> une suite d'expressions à évaluer en séquence. **LETF** va lier la variable fonction en effectuant les 4 étapes suivantes :

1) sauvetage de la valeur de la variable fonction, i.e. la valeur de l'évaluation de (<varfnt>)

2) évaluation de (<varfnt> <e>)

3) évaluation de la séquence <e1> ... <eN>, la valeur de **LETF** sera la valeur de l'évaluation de <eN>

4) restauration de l'ancienne valeur de la variable fonction en évaluant (<varfnt> la valeur calculée en 1).

Ce mécanisme est le même que celui de la fonction **LET** (voir le paragraphe 4.2) pour les variables et c'est ce qui a donné le nom **LETF**.

La réalisation de cette liaison nécessite la création dans la pile d'un nouveau type de bloc de contrôle (le type 2) qui comprend :

{1}		**UNBIND:**
{2}	**PBIND →**	**2**
{3}		**PBIND du bloc précédent**
{4}		**nom de la variable fonction**
{5}		**(son ancienne valeur)**

{1} **UNBIND:** est l'adresse mémoire du module qui réalise la destruction du bloc de contrôle.

{2} **PBIND** pointe sur le type du bloc qui est ici le type 2.

{3} comme tous les blocs de contrôle celui-ci contient le pointeur sur le bloc de contrôle précédent.

{4} le bloc contient le nom de la variable fonction

{5} ainsi que l'ancienne valeur de la variable fonction, sous la forme d'une liste d'un élément.

Ce bloc doit être réalisé par la fonction **LETF** doit voici l'écriture en VCMC2 :

```
; LETF suppose qu'à l'appel :
; A1 <- ((<varfnt> <e>) <e1> ... <eN>)

; Sauvegarde de l'évaluation de ( <varfnt> )

LETF:  CAR     A1,A2
       CDR     A2,TST
       CDR     A1,TST
       CAR     A2,TST
       CAR     A2,A2
       MOVE    NIL,A1,[CALL (EVALFU)]

; Appel de la variable fonction
; avec l'argument fourni

       XCONS   NIL,A1
       POP     A2
       POP     A3
       XTOPST  A1
       PUSH    A2
       PUSH    A3,,[CALL (EVALFU)]

; Fin de préparation du bloc de contrôle
; et exécution du corps du LETF

       POP     A1
       PUSH    (@ PBIND)
       PUSH    '2
       STACK   (@ PBIND)
       PUSH    (UNBIND),,[JUMP (PROGN)]
```

La destruction de ce type de bloc est toutefois spéciale car il faut appeler une nouvelle fois la variable fonction avec l'ancienne valeur sauvée dans le bloc de contrôle. Un problème se pose pour évaluer une dernière fois cette fonction. En effet, l'ancienne valeur de la variable fonction a été sauvée dans le bloc de contrôle mais une simple re-évaluation de la liste composée du nom de la fonction et de la valeur sauvée provoquera (si la variable fonction est de type SUBR ou EXPR) une deuxième évaluation de l'argument.

Pour éviter ce phénomène, nous utiliserons à la place de **EVAL** le module **APPLY** *{Note 1}* qui reçoit dans **A1** le nom d'une fonction et dans **A2** une liste d'arguments prêts à être employés par la fonction.

Voici comment délier un bloc de type 2 :

```
UNBIN2:
        POP     (@ PBIND)       ; restaure l'ancien PBIND
        POP     A4              ; A4 ← le nom
        XTOPST  A3              ; adre ret ↔ (val)
        PUSH    A2              ; sauve tout (APPLY)
        PUSH    A1              ; sauve la valeur
        MOVE    A3,A2           ; A2 ←  la (val)
        MOVE    A4,A1,[CALL (APPLY)] ; Avanti
        POP     A1              ; retourne la valeur du corps
        POP     A2,,[RETURN]    ; libere A2 et rentre
```

{Note 1} ce module qui était le module principal d'évaluation des fonctions en LISP 1.5 a presque disparu dans notre modèle (il est avantageusement remplacé par les modules **EVEXP**, **EVFEXP**, **EVMAC** *et* **EVESC** *de* **EVAL** *qui ne consomment pas de doublets de liste) et n'est plus utilisé que dans le cas des fonctionnelles et du* **LETF**.

7.7 LA NOTION DE CHRONOLOGIE.

Pour résoudre le problème des traces et des pas-à-pas, nous avons inclus dans notre modèle un dispositif général permettant de solutionner les problèmes d'interruptions internes de l'évaluateur (cette étude ne traitera pas des interruptions externes).

Pour cela notre modèle dispose d'une multitude d'évaluateurs potentiels différenciés par un numéro d'ordre de création, un numéro de chronologie.

Ces évaluateurs vont être appelés :

- par l'utilisateur au moyen de la fonction interprète EVAL dont le deuxième argument est le numéro de CHRONOLOGIE que l'on veut voir affecter à cette évaluation :

```
(EVAL  expression  chronologie)
```

- automatiquement par l'interprète, il s'agit alors d'interruptions internes.
  Ces interruptions sont déclenchées par l'interprète lui-même dans les cas suivants :
  - appel de la boucle principale du système **(TOPLEVEL)**
  - erreur à l'interprétation **(ERROR)**
  - trace de l'évaluateur **(STEPEVAL)**
  - ligne pleine en sortie **(EOL)**
  - apparition d'une fin de fichier d'entrée **(EOF)**

  et d'une manière générale à chaque fois que l'interprète doit effectuer quelque chose de non prévu.

Le traitement d'une interruption se fait par invocation d'une fonction propre à chaque type d'interruption, dans un nouvel évaluateur dont la chronologie est la chronologie précédente incrémentée de une unité. Afin de restaurer l'ancienne valeur de la CHRONOLOGIE, le changement d'évaluateur va construire un bloc de contrôle d'un type nouveau, le type **3** qui va contenir l'ancienne valeur de la chronologie, la variable **EVALCH**, ainsi que l'état d'un autre indicateur système **EVALST** (cet indicateur qui contient l'état courant de la trace sera vue au paragraphe 7.7.4.).

Voici la structure de ce nouveau type de bloc de contrôle :

```
        UNBIND:
           3
  ancien  PBIND              <---  PBIND
  ancien  EVALCH
  ancien  EVALST
```

Voici le texte en VCMC2 du traitement d'une interruption interne :

```
    ; Lancement d'une interruption interne
    ; A1 <- la fonction
    ; A2 <- la liste d'arguments (à la APPLY)
    ; change de CHRONOLOGIE  +1

ITSOFT:                   ; prépare un bloc 3
    PUSH   (@ EVALST)     ; sauve l'état de la trace courante
    PUSH   (@ EVALCH)     ; sauve la chronologie courante
    PUSH   (@ PBIND)      ; sauve l'ancien PBIND
    PUSH   '3             ; type de bloc = 3
    STACK  (@ PBIND)      ; new PBIND
    ADD    '1,(@ EVALCH)  ; change de CHRONOLGIE
    PUSH   (UNBIND),,[JUMP (APPLY)] ; et appel APPLY
```

La valeur retournée par la fonction associée à l'interruption devient la valeur de l'interruption.

7.7.1 Fonctions manipulant les CHRONOLOGIES

Notre modèle possède plusieurs fonctions de manipulation de CHRONOLOGIES. Le numéro de la CHRONOLOGIE courante est accessible au moyen de la variable fonction **CHRONOLOGY**.

```
(CHRONOLOGY) lecture      (CHRONOLOGY <n>) écriture
```

Deux fonctions permettent d'utiliser ces CHRONOLOGIES comme des super-structures de contrôle :

```
(EXITCHRONOLOGY <e1> ... <eN>)
(FINDCHRONOLOGY <n> <e1> ... <eN>)
```

La première permet de sortir de la CHRONOLOGIE courante et fait office de retour d'interruption interne (sous la forme d'un super-EXIT), et la seconde permet de retourner une valeur à la CHRONOLOGIE de numéro **<n>**. Dans ces 2 cas la valeur retournée est la valeur de la dernière évaluation i.e. celle de **<eN>** qui a lieu dans l'environnement de l'appel de ces fonctions.

7.7.2 La boucle principale de l'évaluateur

La première interruption de l'évaluateur est engendrée par la boucle principale de l'interprète.

En effet cette boucle est réalisée au moyen des 4 instructions suivantes :

```
MAIN1:
  MOVE   '0,(@ EVALCH)           ; CHRONOLOGIE = 0
  MOVE   'TOPLEVEL,A1            ; le nom de la fonction
  MOVE   NIL,A2,[CALL (ITSOFT)] ; la liste d'arguments
  NOP    ,,[JUMP (MAIN1)]       ; retourne à l'interprète
```

Il est donc tout à fait possible de redéfinir la fonction **TOPLEVEL**.

Classiquement cette fonction réalise :

```
(DE TOPLEVEL ()
    (PRINT (EVAL (READ))))
```

7.7.3 Les erreurs

En cas de détection d'érreur l'évaluateur appelle la fonction **ERROR** en passant par le dispositif d'interruption interne (i.e. l'évaluation de cette fonction a lieu dans un évaluateur dont la CHRONOLOGIE a été incrémentée). Le premier argument de cette fonction contient le type de l'erreur et les suivants des arguments spécifiques dépendant du type de cette erreur. La valeur retournée par cette fonction devient la valeur de l'expression ayant provoquée l'erreur.

Le premier exemple que nous allons voir est une fonction qui évalue son argument sans jamais provoquer d'erreur :

```
(DE EVAL-sans-erreur (forme)
   (WHERE (ERROR) (EVAL forme)))
```

En cas d'erreur dans l'évaluation de la forme, cette fonction retourne NIL. L'utilisation de la forme WHERE permet d'empécher le traitement normal des erreurs uniquement durant l'évaluation de la forme **forme**.

Voici un autre exemple de la redéfinition de la fonction **ERROR** qui permet de construire une fonction testant si une forme peut être évaluée sans erreur par EVAL. Dans le cas ou EVAL produirait une erreur cette fonction retourne la valeur NIL, et dans le cas contraire elle retourne la valeur de l'évaluation. Cette dernière valeur est incluse en premier élément d'une liste pour pouvoir distinguer la valeur NIL correspondant à une évaluation de cette même valeur indiquant une erreur à l'évaluation.

```
(DE EVAL-teste-si-erreur (forme)
   (WHERE (ERROR '(() (erreur)))
      (ESCAPE erreur [(EVAL forme)])))
```

7.7.4 Les traces

Notre modèle possède un mécanisme interne de trace. Ce mécanisme appelle la fonction **STEPEVAL** (qui est une interruption interne) avec comme argument la forme qui devait être évaluée, à chaque appel interne de l'évaluateur. La mode trace est représenté dans le système par une variable interne, **EVALST**.

Ce mode trace est activé en donnant une valeur non-NIL au 3ème argument de la fonction interprète EVAL :

```
(EVAL  expression  chronologie  trace)
```

A l'évaluation d'un appel de ce type, la trace n'est pas active pour la première évaluation de **expression** sous peine de faire boucler EVAL *{Note 1}* mais uniquement pour tous les appels internes qui se produiront durant l'évaluation de **expression**.

La trace la plus simple consiste simplement à imprimer tout ce qui est évalué par EVAL. Voici le texte de cette mini-trace :

```
(DE MINITRACE (exp)
   (WHERE
      (STEPEVAL '((forme)
          (PRINT '↑ forme)
          (EVAL forme  () T)))
      (STEPEVAL exp)))

; exemple d'utilisation

  ?  (MINITRACE '(CONS (CAR '(A)) (CDR '(A B))))
   ↑    (CONS (CAR '(A)) (CDR '(A B)))
   ↑    (CAR '(A))
   ↑    '(A)
   ↑    (CDR '(A B))
```

{Note 1} ce mécanisme est donc équivalent à celui rencontré dans les machines disposant d'interruptions : l'effet des instructions EI *(validation d'interruption) de l'Intel 8080 ou bien* RTI *(retour du sous-programme de trace) du PDP/11, ne se fera sentir qu'après exécution de l'instruction suivante pour éviter de faire boucler l'unité centrale.*

```
        ↑    '(A B)
   =   (A B)
```

Notre modèle prend en compte les interférences possibles entre les fonctions traçantes et les fonctions tracées : en effet les fonctions de type **EXIT** ou **SELF** deviennent ambigues car l'interprète ne sait pas de quelle fonction il s'agit, de la traçante ou de la tracée. Pour résoudre ce problème, nous avons du préciser la définition des fonctions **EXIT** et **SELF** de la manière suivante :
la fonction **EXIT** retourne de la dernière fonction appelée dans la CHRONOLOGIE courante et la fonction **SELF** reapplique la dernière fonction appelée également dans la CHRONOLOGIE courante.

Comme les fonctions traçantes et tracées sont exécutées dans des CHRONOLOGIES différentes, toute ambiguité est donc levée.

Voici une nouvelle fonction de trace qui permet de tracer des fonctions contenant des appels de **SELF** ou de **EXIT**. Cette fonction de plus imprime à chaque étape le numéro de la CHRONOLOGIE courante et la profondeur des appels de EVAL :

```
(DE NTRACEVAL (exp)
   (LET (deep 0)
      (WHERE
         (STEPEVAL '((forme)
            (SETQ deep (ADD1 deep))
            (PRINT '—→ (CHRONOLOGY) deep forme)
            (SETQ IT (EVAL forme (SUB1 (CHRONOLOGY)) T))
            (PRINT '←— (CHRONOLOGY) deep IT)
            (SETQ deep (SUB1 deep))
            IT))
      (EVAL '(STEPEVAL exp) (ADD1 (CHRONOLOGY)))))))
```

La fonction suivante **TF** est une illustration du fonctionnement de cette trace :

```
(DE TF (L) (IF (NULL L) (EXIT 'OK) (SELF (CDR L)))))

 ?  (NTRACEVAL '(TF '(A B)))
   →    1    1    (TF '(A B))
   →    1    2    '(A B)
   ←    1    2    (A B)
   →    1    2    (IF (NULL L) (EXIT 'OK) (SELF (CDR L)))
```

```
        →   1   3   (NULL L)
        →   1   4   L
        ←   1   4   (A B)
        ←   1   3   NIL
        →   1   3   (SELF (CDR L))
        →   1   4   (CDR L)
        →   1   5   L
        ←   1   5   (A B)
        ←   1   4   (B)
        →   1   4   (IF (NULL L) (EXIT 'OK) (SELF (CDR L)))
        →   1   5   (NULL L)
        →   1   6   L
        ←   1   6   (B)
        ←   1   5   NIL
        →   1   5   (SELF (CDR L))
        →   1   6   (CDR L)
        →   1   7   L
        ←   1   7   (B)
        ←   1   6   NIL
        →   1   6   (IF (NULL L) (EXIT 'OK) (SELF (CDR L)))
        →   1   7   (NULL L)
        →   1   8   L
        ←   1   8   NIL
        ←   1   7   T
        →   1   7   (EXIT 'OK)
        ←   1   7   OK
        ←   1   6   OK
        ←   1   5   OK
        ←   1   4   OK
        ←   1   3   OK
        ←   1   2   OK
        ←   1   1   OK
      = OK
```

Enfin le dispositf de trace de notre modèle permet d'implémenter des dispositifs d'évaluation incrémentale. La fonction **STEP** décrite ci-dessous va évaluer une expression mais avant et après chacune des évaluations internes, le contrôle est rendu à l'opérateur qui peut indiquer à chaque étape des actions à entreprendre. Ces actions, décrites par un caractère, peuvent être :

- v : voir l'expression à évaluer (cette action est automatique à chaque étape).
- > : continuer la trace en séquence.
- < : évaluer l'expression courante sans trace et se remettre en mode trace à la fin de l'évaluation de celle-ci.
- = : lire et évaluer une expression quelconque (ce qui permet de contrôler les valeurs des variables ou d'évaluer n'importe quelle forme) et repasser en mode trace.
- . : arrêter le mode trace et retourner la valeur de l'expression tracée.

```
(DE STEP (exp)
  (LET ((deep 0) (typit T) (IT NIL))
    (WHERE
      (STEPEVAL '((forme)
         (SETQ deep (ADD1 deep))
         (LET (cmd  'v)
            (IF typit
               (SELECTQ cmd
                  (v (PRIN '-> deep forme '||)
                     (TERPRI -1)
                     (SELF (READCH)))
                  (> (SETQ IT (EVAL forme (SUB1 (CHRONOLOGY)) T)))
                  (< (SETQ IT (EVAL forme (SUB1 (CHRONOLOGY)) NIL)))
                  (= (PRINT (EVAL (READ))) (SELF 'v))
                  (/. (SETQ typit ())
                      (EVAL forme (SUB1 (CHRONOLOGY))))
                  (T (SELF 'v)))
               (SETQ IT (EVAL forme (SUB1 (CHRONOLOGY))))))
         (LET (cmd 'v)
            (IF typit
               (SELECTQ cmd
                  (v (PRIN '<- deep IT '||)
                     (TERPRI -1)
                     (SELF (READCH)))
                  (= (PRINT (EVAL (READ) (ADD1 (CHRONOLOGY))))
                     (SELF 'v))
                  (/. (SETQ typit))
                  (< )
                  ((SELF 'v)))))
         (SETQ deep (SUB1 deep))
         IT))
      (EVAL '(STEPEVAL exp) (ADD1 (CHRONOLOGY))))))))))
```

Le fait de pouvoir récupérer TOUS les appels internes de l'évaluateur permet, outre les traces, une modification complète de celui-ci.

Voici pour illustration la fonction **EVAL-NIL-si-UNDEF** qui se comporte comme la fonction interprète standard EVAL mais qui ne provoque pas d'erreur à l'occurrence de l'évaluation d'une variable indéfinie, se contentant uniquement de forcer la valeur NIL à la variable et d'imprimer un message d'avertissement :

```
(DE EVAL-NIL-si-UNDEF (expression)
    (WHERE
       (STEPEVAL '((forme)
          (IF (OR (LISTP forme) (NUMBP forme) (BOUNDP forme))
              (EVAL forme () T)
              (PRINT "Je donne la valeur NIL à :" forme)
              (SET forme NIL))))
       (STEPEVAL expression)))
```

Voici quelques utilisations de cette fonction en supposant que les variables VARF001, VARF002 et VARF003 sont indéfinies

```
?    (EVAL-NIL-si-UNDEF 'VARF001)
        Je donne la valeur NIL à : VARF001
☞   NIL

?    (EVAL-NIL-si-UNDEF '['A VARF002 'B VARF003])
        Je donne la valeur NIL à : VARF002
        Je donne la valeur NIL à : VARF003
☞   (A NIL B NIL)
```

On notera là encore la redéfinition dynamique de la fonction **STEPEVAL**, qui permet de n'effectuer le test de variable indéfinie que dans la portée dynamique de la fonction **EVAL-NIL-si-UNDEF**.

7.8 ANALYSE DE L'INTERPRETE.

Enfin voici le tableau de passages dans les différents modules de l'interprète (donné à l'appendice F) après avoir effectué le test (donné à l'appendice G). La première colonne de pourcentages est donnée par rapport au nombre d'appels de l'évaluateur, et la seconde est donnée par rapport au nombre total d'instructions exécutées durant l'évaluation du test.

Ce tableau nous a permis d'établir les fréquences d'utilisation des différents modules de EVAL données au paragraphe 4.4.

Passage aux étiquettes

Nb de passages dans EVALA1 : 4647
Nb d'instructions exécutées : 80202

1	ADD1	= 57	1.23 %	0 %
2	APPEX1	= 28	0.6 %	0 %
3	APPEX2	= 41	0.88 %	0 %
4	APPEX3	= 13	0.28 %	0 %
5	APPEXP	= 13	0.28 %	0 %
6	APPLIN	= 114	2.45 %	0.14 %
7	APPLY	= 124	2.67 %	0.15 %
8	APPLY2	= 1	0 %	0 %
9	APPLYL	= 10	0.22 %	0 %
10	APPLYS	= 10	0.22 %	0 %
11	CADR	= 8	0.17 %	0 %
12	CAR	= 210	4.52 %	0.26 %
13	CDDR	= 5	0.11 %	0 %
14	CDR	= 451	9.71 %	0.56 %
15	COND	= 94	2.02 %	0.12 %
16	COND1	= 207	4.45 %	0.26 %
17	CONS	= 140	3.01 %	0.17 %
18	DE	= 19	0.41 %	0 %
19	DEF	= 22	0.47 %	0 %
20	DF	= 1	0 %	0 %
21	DIFFER	= 33	0.71 %	0 %
22	DM	= 2	0 %	0 %
23	EPROGN	= 252	5.42 %	0.31 %
24	EQ	= 206	4.43 %	0.26 %
25	ERR02	= 9	0.19 %	0 %
26	ERR04	= 14	0.3 %	0 %
27	ERROR	= 5	0.11 %	0 %
28	ESCAPE	= 25	0.54 %	0 %
29	EVAL	= 2	0 %	0 %
30	EVAL0	= 1	0 %	0 %
31	EVAL1	= 1052	22.64 %	1.31 %
32	EVAL2	= 493	10.61 %	0.61 %
33	EVAL3	= 1	0 %	0 %
34	EVALA1	= 4647	100. %	5.79 %
35	EVALAN	= 4649	100.0 %	5.8 %
36	EVALAT	= 1591	34.24 %	1.98 %
37	EVALF	= 721	15.52 %	0.9 %
38	EVALFAT	= 2760	59.39 %	3.44 %
39	EVALFLI	= 10	0.22 %	0 %
40	EVALFNB	= 1	0 %	0 %
41	EVALFU	= 2771	59.63 %	3.46 %
42	EVALI	= 3	0 %	0 %
43	EVALIN	= 2763	59.46 %	3.45 %
44	EVALIS	= 2763	59.46 %	3.45 %

45	EVALL	=	5	0.11	%	0	%
46	EVALL2	=	1	0	%	0	%
47	EVALL3	=	1	0	%	0	%
48	EVALN	=	3	0	%	0	%
49	EVALN1	=	3	0	%	0	%
50	EVALN2	=	4	0	%	0	%
51	EVALNB	=	295	6.35	%	0.37	%
52	EVCAR	=	4276	92.02	%	5.33	%
53	EVESC	=	3	0	%	0	%
54	EVESC1	=	4	0	%	0	%
55	EVESC3	=	3	0	%	0	%
56	EVESC4	=	3	0	%	0	%
57	EVEXP	=	493	10.61	%	0.61	%
58	EVEXP1	=	654	14.07	%	0.82	%
59	EVEXP2	=	1147	24.68	%	1.43	%
60	EVEXP3	=	493	10.61	%	0.61	%
61	EVEXP4	=	656	14.12	%	0.82	%
62	EVEXP5	=	1149	24.73	%	1.43	%
63	EVEXP6	=	649	13.97	%	0.81	%
64	EVEXP9	=	155	3.34	%	0.19	%
65	EVEXPF	=	361	7.77	%	0.45	%
66	EVEXPG	=	361	7.77	%	0.45	%
67	EVEXPN	=	338	7.27	%	0.42	%
68	EVFEX2	=	10	0.22	%	0	%
69	EVFEX3	=	20	0.43	%	0	%
70	EVFEXB	=	10	0.22	%	0	%
71	EVFEXP	=	2	0	%	0	%
72	EVLIS	=	12	0.26	%	0	%
73	EVLIS1	=	39	0.84	%	0	%
74	EVLIS2	=	12	0.26	%	0	%
75	EVMAC	=	8	0.17	%	0	%
76	EXICH1	=	5	0.11	%	0	%
77	EXICH2	=	5	0.11	%	0	%
78	EXIT	=	1	0	%	0	%
79	EXIT1	=	2	0	%	0	%
80	EXIT2	=	1	0	%	0	%
81	EXITCHRON	=	5	0.11	%	0	%
82	EXITER	=	1	0	%	0	%
83	FALSE	=	12	0.26	%	0	%
84	FVAL	=	1	0	%	0	%
85	FVAL1	=	1	0	%	0	%
86	IF	=	385	8.28	%	0.48	%
87	INTERNAL	=	3	0	%	0	%
88	ITSOFT	=	108	2.32	%	0.13	%
89	LAMBDA	=	3	0	%	0	%
90	LET	=	4	0	%	0	%
91	LET2	=	6	0.13	%	0	%
92	LET4	=	2	0	%	0	%
93	LET8	=	4	0	%	0	%
94	LETF	=	3	0	%	0	%
95	LIST	=	10	0.22	%	0	%
96	LT	=	23	0.49	%	0	%
97	MAIN	=	1	0	%	0	%
98	MAIN1	=	103	2.22	%	0.13	%
99	MAP	=	2	0	%	0	%
100	MAP1	=	6	0.13	%	0	%
101	MAP11	=	8	0.17	%	0	%
102	MAP2	=	9	0.19	%	0	%
103	MAP3	=	28	0.6	%	0	%
104	MAP4	=	16	0.34	%	0	%
105	MAP5	=	28	0.6	%	0	%
106	MAP8	=	37	0.8	%	0	%
107	MAPC	=	1	0	%	0	%
108	MAPC1	=	4	0	%	0	%
109	MAPC11	=	5	0.11	%	0	%

110	MAPC2	=	4	0	%	0	%
111	MAPC3	=	16	0.34	%	0	%
112	MAPC4	=	6	0.13	%	0	%
113	MAPC5	=	16	0.34	%	0	%
114	MAPC6	=	20	0.43	%	0	%
115	MEMQ	=	63	1.36	%	0	%
116	MEMQ1	=	43	0.93	%	0	%
117	MEMQ2	=	37	0.8	%	0	%
118	NOT	=	251	5.4	%	0.31	%
119	NULL	=	251	5.4	%	0.31	%
120	OUTLIN	=	498	10.72	%	0.62	%
121	PLUS	=	33	0.71	%	0	%
122	POPJ	=	599	12.89	%	0.75	%
123	PRIN	=	174	3.74	%	0.22	%
124	PRIN11	=	100	2.15	%	0.12	%
125	PRININI	=	1	0	%	0	%
126	PRINT	=	72	1.55	%	0	%
127	PROBJ	=	9	0.19	%	0	%
128	PROBJT	=	209	4.5	%	0.26	%
129	PROGN	=	863	18.57	%	1.08	%
130	PROGNA3	=	252	5.42	%	0.31	%
131	QUOTE	=	78	1.68	%	0	%
132	READINI	=	1	0	%	0	%
133	REV1	=	23	0.49	%	0	%
134	REVERSE	=	32	0.69	%	0	%
135	RPLACB	=	4	0	%	0	%
136	SELF	=	11	0.24	%	0	%
137	SELF1	=	21	0.45	%	0	%
138	SELFER	=	1	0	%	0	%
139	SELFF	=	9	0.19	%	0	%
140	SELFL	=	10	0.22	%	0	%
141	SELFS	=	1	0	%	0	%
142	SELFW	=	9	0.19	%	0	%
143	SERR01	=	14	0.3	%	0	%
144	SERR02	=	5	0.11	%	0	%
145	SERROR	=	5	0.11	%	0	%
146	SETQ	=	11	0.24	%	0	%
147	STOP	=	1	0	%	0	%
148	SUB1	=	75	1.61	%	0	%
149	TEVEXL	=	280	6.03	%	0.35	%
150	TEVEXW	=	31	0.67	%	0	%
151	TIMES	=	5	0.11	%	0	%
152	TOPLEVEL	=	103	2.22	%	0.13	%
153	TRUE	=	227	4.88	%	0.28	%
154	UDFA	=	1	0	%	0	%
155	UDFA1	=	1	0	%	0	%
156	UDFE	=	1	0	%	0	%
157	UDFER	=	1	0	%	0	%
158	UNBDF	=	3	0	%	0	%
159	UNBDL	=	361	7.77	%	0.45	%
160	UNBDL1	=	406	8.74	%	0.51	%
161	UNBDL2	=	767	16.51	%	0.96	%
162	UNBDS	=	108	2.32	%	0.13	%
163	UNBDW	=	28	0.6	%	0	%
164	UNBIND	=	490	10.54	%	0.61	%
165	UNBINP	=	500	10.76	%	0.62	%
166	WHERE	=	3	0	%	0	%
167	WHERE1	=	2	0	%	0	%
168	WHERE2	=	3	0	%	0	%
169	WHERE3	=	28	0.6	%	0	%

BIBLIOGRAPHIE

[ALLEN 78]
ALLEN J. : *The Anatomy of LISP,* Mc Graw Hill Inc., 1978.

[AMD 78a]
The Am2900 Family Data Book, Advanced Micro Devices Inc., 1978.

[AMD 78b]
Build a Microcomputer, Chapter II, Microprogrammed Design, Advanced Micro Devices Inc., AM-PUB073-2, 1978.

[AMD 78c]
PARKER R. O. & KROEGER J. H. : *Algorithm Details for the Am9511 Arithmetic Processing Unit,* Advanced Micro Devices Inc., AM-PUB072, 1978.

[AMD 79]
AmZ8000 Family Reference Manual, *Principles of Operation & AmZ8001/2 Processor Instruction Set,* Advanced Micro Devices Inc., AM-PUB086, 1978.

[ARCHIBALD 77]
ARCHIBALD P. : *FONTS MANUAL,* IBM Thomas J. Watson Research Center, Yorktown Heights, N.Y., Version 2, September 1977.

[AUDOIRE 76]
AUDOIRE L. : *COLORIX : un périphérique de visualisation couleur,* Mémoire de maîtrise, UER Informatique, Université de Paris 8 - Vincennes, Juin 1976.

[BAKER 77a]
BAKER H. G. Jr. : *Shallow binding in LISP 1.5,* M.I.T. Artificial Intelligence Laboratory, Working Paper 138, January 1977 and C.A.C.M., Vol. 21 No .7 pp. 565-569, July 1978.

[BAKER 77b]
BAKER H. G. Jr. : *A Note on the Optimal Allocation of Spaces in MACLISP,* M.I.T. Artificial Intelligence Laboratory, Working Paper 142, March 1977.

[BAKER 78]
BAKER H. G. Jr. : *List Processing in Real Time on a serial Computer,* C.A.C.M., Vol. 21 No. 4 pp. 280-294, April 1978.

[BERKELEY 74]
BERKELEY E. C., BOBROW D. G. (editors) : *The Programming Langage LISP : Its Operation and Application,* The M.I.T. Press, Cambridge, Massachusets, 4th printing, March 1974.

[BOBROW 73a]
BOBROW R. J., BURTON R. R., JACOBS J. M., LEWIS D. : *UCI LISP Manual,* Dept. of Information and Computer Science, University of California, Irvine, Technical Report #21 updated 10/73.

[BOBROW 73b]
BOBROW D. G., WEGBREIT B. : *A Model and Stack Implementation of Multiple Environments,* C.A.C.M., Vol. 16 No. 10 pp. 591-603, October 1973.

[BOLCE 68]
BOLCE J. F. : *LISP/360 : a Description of the University of WATERLOO LISP 1.5 for the IBM System 360,* 2nd ed., University of WATERLOO, Computer Centre, March 1968.

[BURSTALL 71]
BURSTALL R. M., COLLINS J. S., POPPLESTONE R. J., *Programming in POP 2,* Edinburg University Press, 1971.

[CARPENTER 76]
CARPENTER B. E., DORAN R. W. : *The Other Turing Machine,* The Computer Journal, Vol. 20 No. 3 pp. 269-279, March 1976.

[CHAILLOUX 78a]
CHAILLOUX J. : *VLISP 8 un système LISP pour micro-processeur à mots de 8 bits,* RT 21-78, Département d'Informatique, Université de Paris 8 - Vincennes, Juillet 1978.

[CHAILLOUX 78b]
CHAILLOUX J. : *a VLISP interpreter on the VCMC1 machine,* LISP Bulletin #2, July 1978.

[CHAILLOUX 78c]
CHAILLOUX J. : VLISP *10 . 3 , Manuel de Référence,* RT 16-78, Université de Paris 8 - Vincennes, Août 1978.

[CHAILLOUX 79a]
CHAILLOUX J. : VLISP *8 . 2 , Manuel de Référence,* RT 11-79, Université de Paris 8 - Vincennes, Avril 1979.

[CHAILLOUX 79b]
CHAILLOUX J. : *Etude d'un compilateur* VLISP *optimisant*, Bulletin du Groupe Programmation et Langages, A.F.C.E.T., Division Théorique et Technique de l'Informatique, No. 9 pp. 26-36, Octobre 1979.

[CLARK 77]
CLARK D. W. : *A empirical study of list structure in LISP*, C.A.C.M., Vol. 20 No. 2 pp. 78-87, February 1977.

[CLARK 79]
CLARK D. W. : *Measurement of Dynamic List Structure Use in LISP*, IEEE Soft. Eng. , Vol. SE-5 No. 1 pp. 51-59, January 1979.

[COILLAND 79]
COILLAND P., COLAITIS M. J. : *SIRLISP : Interprète LISP sur IRIS80*, E.N.S.T., Paris, Février 1979.

[DEC 75a]
PDP 11 : Processor Handbook, Digital Equipment Corporation, Maynard Massachusetts, 1975.

[DEC 75b]
DECSYSTEM 10 : *Getting Started With RUNOFF, Text Formatting Program*, Digital Equipment Corporation, DEC-10-URUNA-A-D, Maynard Massachusetts, February 1975.

[DEC 75c]
DECSYSTEM 10 : *DDT, Dynamic Debbuging Technique*, Digital Equipment Corporation, DEC-10-UDDTA-A-D, Maynard Massachusetts, 1975.

[DEC 78a]
DECSYSTEM10 : *System Reference Manual*, Digital Equipment Corporation, Maynard Massachusetts, AD-09 16C-T1, March 1978.

[DEC 78b]
DECSYSTEM10 : *TOPS10*, Digital Equipment Corporation, Maynard Massachusetts, AD-09 16C-T1, March 1978.

[DEC 78c]
RT11 V03 : *Documentation directory*, Digital Equipment Corporation, Order No. DEC 11-ORDDB-A-D, Maynard Massachusetts, 1978.

[DEC 78d]
LSI 11 : *Microprocessor Handbook*, Digital Equipment Corporation, Maynard Massachusetts, 1978.

[DEC 78e]
VAX 11-780, Hardware/Software Handbook, Digital Equipment Corporation, Maynard Massachusetts, 1978.

[DEUTSCH 73]
DEUTSCH L. P. : *A LISP Machine with Very Compact Programs*, Proc. 3rd I.J.C.A.I., pp. 697-703, Stanford, California, August 1973.

[DEUTSCH 76]
DEUTSCH L. P., BOBROW D. G. : *An Efficient, Incremental, Automatic Garbage Collector*, C.A.C.M., Vol. 19 No. 9 pp. 522-526, September 1976.

[DIJKSTRA 78]
DIJKSTRA E. W., LAMPORT L., MARTIN A. J., SCHOLTEN C. S., STEFFENS E. F. M. : *On-the-Fly Garbage Collection : An Exercice in Cooperation*, C.A.C.M., Vol. 21 No. 11 pp. 966-975, November 1978.

[DURIEUX 78]
DURIEUX J. L. : *TLISP, le système LISP de TOULOUSE*, in Implémentation et Interprétation de LISP, Ecole IRIA, Toulouse, 1-3 Mars 1978.

[DURIEUX 79]
DURIEUX J. L., VIGNOLLE J. : *Fausses Récursivités dans un Interprète LISP : une approche formelle de leur élimination*, Université Paul Sabatier, Toulouse.

[EARNEST 76]
EARNEST L. : *FIND a FONT*, Stanford Artificial Intelligence Laboratory, Operating Note 74, May 1976.

[FROST 75]
FROST M. : *UUO Manual (Second Edition)*, SAILON 55.4, Stanford Artificial Intelligence Laboratory, July 1975.

[FROST 77]
FROST M. & HARVEY B. : *The Stanford/IRCAM Monitor*, Institut de Recherche et Coordination Acoustique/Musique, No. 2, October 1977.

[GARDNER 67]
GARDNER M. : *Mathematical Games*, Scientific American, pp. 124-125, March/April 1967.

[GOLDSTEIN 73]
GOLDSTEIN I. : *Pretty Printing : converting List to Linear Structure*, M.I.T. Artificial Intelligence Laboratory, AI memo No. 279, February 1973.

[GOOSSENS 77]
GOOSSENS D. : *CAN*, Département d'Informatique, Université de Paris 8 - Vincennes, 1977.

[GOOSSENS 79]
GOOSSENS D. : *Meta-interpretation of Recursive List-processing Programs*, Proc. 6th I.J.C.A.I. pp. S7-S12, Tokyo, August 1979.

[GREENBLATT 74]
GREENBLATT R. : *The LISP Machine*, M.I.T. Artificial Intelligence Laboratory, Working Paper 79, November 1974.

[GREUSSAY 72]
GREUSSAY P. : *Manuel LISP 510 : description et utilisation*, Institut d'Intelligence Artificielle, Université de Paris 8 - Vincennes, NTP 2, Octobre 1972.

[GREUSSAY 75]
GREUSSAY P. : *LISP T1600 : Manuel de Référence*, Département d'Informatique, Université de Paris 8 - Vincennes, Février 1975.

[GREUSSAY 76a]
GREUSSAY P. : *VLISP : Structure et extension d'un système LISP pour mini-ordinateur*, RT 16-76, UER Informatique et Linguistique, Université de Paris 8 - Vincennes, Janvier 1976.

[GREUSSAY 76b]
GREUSSAY P. : *Descriptions compactes d'interprètes implémentables : une application au langage CONNIVER*, 2ème Colloque International sur la Programmation, ROBINET B. (éd.), Dunod, Paris, Avril 1976.

[GREUSSAY 76c]
GREUSSAY P. : *Iterative interpretations of tail-recursive LISP procedures*, Département d'Informatique, TR 20-76, Université de Paris 8 - Vincennes, Septembre 1976.

[GREUSSAY 77]
GREUSSAY P. : *Contribution à la définition interprétative et à l'implémentation des LAMBDA-langages*, Thèse, Université de Paris VII, Novembre 1977.

[GREUSSAY 78a]
GREUSSAY P. : *Communication personnelle*, Octobre 1978.

[GREUSSAY 78b]
GREUSSAY P. : *Le Système VLISP 16*, Ecole Polytechnique, Décembre 1978.

[GREUSSAY 79a]
GREUSSAY P. : *Aides à la Programmation en LISP : outils d'observation et de compréhension*, Bulletin du Groupe Programmation et Langages, A.F.C.E.T., Division Théorique et Technique de l'Informatique, No. 9 pp. 13-25, Octobre 1979.

[GREUSSAY 79b]
GREUSSAY P. : *VLISP-11 Manuel de Référence, (à paraître),* Université de Paris 8 - Vincennes, 1979.

[GRIGNETTI 76]
GRIGNETTI M. C., HARTLEY A. K., HAUSMANN C., ASH W. L., BOBROW R. J., BELL A. : ***Intelligent On-line Assistant an Tutor System,*** Technical progress Report, Bolt Beranek and Newman Inc., BBN Report No. 3302, AI Report No. 41, March 1976.

[GRISWOLD 71]
GRISWOLD R. E., POAGE J. F., POLONSKY I. P. : ***The SNOBOL4 Programming Language,*** 2nd ed., Prentice Hall, 1971.

[HAFNER 74]
HAFNER C., WILCOX B. : *LISP/MTS programmer's guide,* Mental Health Research Institute, Michigan University, Ann ARBOR, January 1974.

[HAILPERN 79]
HAILPERN B. T., HITSON B. L. : ***S-1 Architecture Manual,*** Computer System Laboratory, Stanford Electronics Laboratories, Stanford University, Technical Report No. 161, STAN-CS-79-715, January 1979.

[HAINES 69]
HAINES E. C. : ***TREET a List Processing Language and System,*** the MITRE corp., MTP104, March 1979.

[HARVEY 74]
HARVEY B. : *Monitor Command Manual (Second Edition),* SAILON 54.4, Stanford Artificial Intelligence Laboratory, September 1974.

[HEARN 69]
HEARN A. C. : ***Standard LISP,*** in Bobrow D. G. (ed.) : LISP Bulletin, ACM SIGPLAN Notices, Vol. 4 no. 9, September 1969.

[HEARN 73]
HEARN A. C. : ***REDUCE 2 User's Manual,*** Utah Computational Physics group UCP-19, Second Ed., March 1973.

[HEARN 74]
HEARN A. C. : ***REDUCE 2 Symbolic Mode Primer,*** Utah Computational Physics group Operating Note 5.1, October 1974.

[HOLLOWAY 80]
HOLLOWAY J., STEELE G., SUSSMAN G., BELL A. : ***The SCHEME 79 Chip,*** M.I.T. Artificial Intelligence Laboratory, AI Memo No. 255, Draft, January 1980.

[HORNING 79]
HORNING J. J. : ***Additionnal Viewpoints on the History of Programming Languages Conference,*** Annals of the History of Computing, Vol. 1 pp. 69-71, July 1979.

[HUITRIC 76]
HUTRIC H. : *Couleur et calcul : calcul de la couleur*, Thèse de 3ème cycle, Université de Paris 8, 1976.

[HULLOT 79]
HULLOT J. M. : *Associative Commutative Pattern-matching*, Proc. 6th I.J.C.A.I. pp. 406-412, Tokyo, August 1979.

[INTEL 77a]
8080 : *Assembly Language Programming Manual*, ON : 98-00301B, Intel Corporation, 1977.

[INTEL 77b]
MDS : *ISIS II System user's guide*, ON : 98-306B, Intel Corporation, 1977.

[INTEL 79]
8086 : *16 bit HMOS Microprocessor*, Intel Corparation, 1979.

[JOUANNAUD 77]
JOUANNAUD J. P. : *Sur l'inférence et la synthèse automatiques de fonctions LISP à partir d'exemples*, Thèse, Université Pierre et Marie Curie (Paris 6), Novembre 77.

[KNIGHT 74]
KNIGHT T. : *The CONS Machine*, M.I.T. Artificial Intelligence Laboratory, Working Paper 80, November 1974.

[KNUTH 69]
KNUTH D. E. : *The Art of Computer Programming, Vol. 1 : Fundamental Algorithms*, Addison-Wesley, Reading, Massachusetts, 2nd printing, 1969.

[KNUTH 78a]
KNUTH D. E. : *Mathematical Typography*, Stanford University, STAN-CS-78-648, February 1978.

[KNUTH 78b]
KNUTH D. E. : *TAU EPSILON CHI : a System for Technical Text*, Stanford Artificial Intelligence Laboratory, Memo AIM-317, STAN-CS-78-675, September 1978.

[KUROKAWA 77]
KUROKAWA T. : *A New Fast and Safe Marking Algorithm*, Informations Systems Lab., TOSHIBA Center, Kawasaki, Japan, April 1977.

[LAUBSCH 76]
LAUBSCH J. H., KRAUSE D., HESS K., SCHATZ W. : *MACLISP Manual*, CUU-Memo-3, Universitat Stuttgart, Stuttgart, Mai 1976

[LECOUFFE 77]
LECOUFFE P. : *étude et définition d'une machine langage LISP*, Thèse de spécialité, Université des sciences et techniques de LILLE, Décembre 1977.

[LECOUFFE 79]
LECOUFFE P. : *Communication et Mémorisation en LISP,* Bulletin du Groupe Programmation et Langages, A.F.C.E.T., Division Théorique et Technique de l'Informatique, No. 9 pp. 37-46, Octobre 1979.

[LISPMACHINE 77]
LISP MACHINE GROUP, BAWDEN A., GREENBLATT R., HOLLOWAY J. KNIGHT T., MOON D., WEINREB D. : *LISP Machine Progress Report,* M.I.T. Artificial Intelligence Laborartory, Memo No. 444, August 1977.

[LUX 78]
LUX A. : *LISP - IRIS 80 : Manuel d'Utilisation,* Laboratoire IMAG, Février 1978.

[MAAS 78]
MAAS R. E., GORIN R. E. : *Prototype Overlay Xerographics,* POX writeup, in file POX.XGP[UP,DOC].

[MACSYMA 75]
MACSYMA Reference Manual, Project MAC Mathlab Group, M.I.T., November 1975.

[MARTI 79]
MARTI J., HEARN A. C., GRISS M. L., GRISS C. : *STANDARD LISP REPORT,* ACM SIGPLAN Notices, Vol. 14 No. 10, October 1979.

[McCARTHY 60a]
McCARTHY J. : *LISP 1 Programmer's manual,* M.I.T. Computation Center & Research Laboratory of Electronics, Cambridge Mass., March 1, 1960.

[McCARTHY 60b]
McCARTHY J. : *Recursive Functions of Symbolic Expressions and their Computation by Machine Part I,* C.A.C.M., vol. 3, March 1960.

[McCARTHY 62]
McCARTHY J., ABRAHAMS P. W., EDWARDS D. J., HART T. P., LEVIN M. I. : *LISP 1.5 Programmer's manual,* the M.I.T. Press, Cambridge, Mass., 1962.

[McCARTHY 78]
McCARTHY J. : *History of LISP,* History of Programming Languages Conference, ACM SIGPLAN Notices, Vol. 13 No. 8, 1978.

[MEAD 80]
MEAD C., CONWAY L. : *Introduction to VLSI systems,* Addison-Wesley publishing company, 1980.

[MODEL 79]
MODEL M. L. : *Monitoring System Behaviour in a Complex Computational Environment,* XEROX, Palo Alto Research Center, CSL-79-1, January 1979.

[MOON 74]
MOON D. A. : *MACLISP Reference Manual,* M.I.T. Project MAC, Cambridge Mass., April 1974.

[MOORE 76]
MOORE J. S. : *the INTERLISP Virtual Machine Specification,* XEROX Palo Alto Research Center, Palo Alto Cal., September 1976.

[MOSES 70]
MOSES J. : *the function of FUNCTION in LISP or Why the FUNARG Problem should be called the Environment Problem,* Memo 199, M.I.T. Artificial Intelligence Laboratory, June 1970.

[MOTOROLA 79]
MOTOROLA : *MC68000,* Motorola Semiconductors, Advance Specification, 1979.

[NAGAO 79]
NAGAO M., TSUJII J., NAKAJIMA K., MITAMURA K., ITO H. : *LISP Machine NK3 and measurement of its performance,* Proc. 6th I.J.C.A.I. pp. 625-627, Tokyo, August 1979.

[NIVAT 79]
NIVAT M. : *La Recherche en Programmation et ses Moyens de Calcul,* La Gazette du L.I.T.P. Bulletin No. 9, Janvier 1979.

[PERROT 79]
PERROT J. F. : *LISP et lambda-calcul,* Lambda-Calcul et Semantique formelle des langages de programmation B. Robinet, Ed., pp. 277-301, LITP-ENSTA, Paris, 1979.

[POOLE 70]
POOLE P. C., WAITE W. M. : *The STAGE2 Macro-processor User Reference Manual,* Culham Laboratories Abingdon, Berks, July 1970.

[PRATT 76]
PRATT V. R. : *CGOL - an Alternative External Representation for LISP users,* M.I.T. Artificial Intelligence Laboratory, Working Paper 121, March 1976.

[QUAM 72]
QUAM L. H., DIFFIE W. : *Stanford LISP 1.6 Manual,* SAILON 28.6, Computer Science Dpt, Stanford University, 1972.

[ROBINET 78]
ROBINET B. : *petit précis de λ calcul,* in Implémentation et Interprétation de LISP, Ecole IRIA, Toulouse, pp. 15-24, 1-3 Mars 1978.

[ROBINET 79]
ROBINET B. : *De la sémantique dénotationnelle,* rapport L.I.T.P. No. 79-3, Janvier 1979.

[ROSEN 72]
ROSEN E. : A User's Guide to the Fast Arithmetic Feature of the LISP compiler, M.I.T., 1972.

[SAMMUEL 77]
SAMUEL A. : *E,* Institut de Recherche et de Coordination Acoustique/Musique, No. 0, Avril 1977.

[SHIMADA 76]
SHIMADA T. ,YAMAGUSHI Y., SAKAMURA K. : *A LISP Machine and Its Evaluation,* Systems Computers Controls, Vol. 7, No. 3, 1976 (translated from Denshi Tsushin Gakkai Rombunshi, Vol. 59-d, No. 6 June 1976, pp. 406-413).

[SCHOETTL 75]
SCHOETTL J. E., WERTZ H. : *Des dragons à foison,* Artinfo/Musinfo #21, Groupe Art et Informatique, Université de Paris 8 - Vincennes, 1975.

[SMITH 73]
SMITH D. C., ENEA H. J. : *MLISP 2,* Artificial Intelligence Memo no. 195, Stanford University, May 1973.

[STEELE 74]
STEELE G. L. Jr. : *Annoucing the One, the Only BIBOP LISP,* M.I.T. Artificial Intelligence Laboratory, March 1974.

[STEELE 75]
STEELE G. L. Jr. : *Multiprocessing Compactifying Garbage Collection,* C.A.C.M., Vol. 18 No. 9 pp. 495-508, September 1975.

[STEELE 77]
STEELE G. L. Jr. : *Macaroni is better than Spaghetti,* Proc. of the Symposium of Artificial Intelligence and Programming Languages, ACM SIGPLAN Notices, Vol. 12 No. 8, August 1977.

[STEELE 79]
STEELE G. L. Jr., SUSSMAN G. J. : *Design of LISP-Based Processors or, SCHEME: A Dielectric LISP or, Finite Memories Considered Harmful or, LAMBDA: The Ultimate Opcode,* AI Memo No. 514, M.I.T. Artificial Intelligence Laboratory, Cambridge, Mass., March 1979.

[STOYAN 78a]
STOYAN H. : *LISP,* Thèse, Technishe Universitat Dresden, Dresden, RDA.

[STOYAN 78b]
STOYAN H. : ***LISP-Programmier Handbuch,*** Akademie Verlag, Berlin, 1978.

[STRACHEY 65]
STRACHEY C. : ***A General Purpose Macrogenerator,*** Computer Journal pp. 225-241, October 1965.

[TAFT 79]
TAFT S. T. : ***The Design of an M6800 LISP Interpreter,*** BYTE, Vol. 4 No. 8 pp. 132-152, August 1979.

[TAKI 79]
TAKI K., KANEDA Y., MAEKEWA S. : ***The Experimental LISP Machine,*** Proc. 6th I.J.C.A.I., TOKYO, 1979.

[TEITELMAN 73]
TEITELMAN W. : ***CLISP - Conversationnal LISP,*** Proc. 3rd I.J.C.A.I., pp. 686-690, Stanford, California, August 1973.

[TEITELMAN 75]
TEITELMAN W. : ***INTERLISP Reference Manual,*** 2nd revision, Xerox Palo Alto Research Center, December 1975, 3rd revision, October 1978.

[TEITELMAN 77]
TEITELMAN W. : ***A Display Oriented Programmer's Assistant,*** Proc. 5th I.J.C.A.I., pp. 905-915,Cambridge, Mass., August 1977.

[TESLER 72]
TESLER L. : ***PUB the Document Compiler,*** Stanford Artificial Intelligence Project, Operating Note 70, September 1972.

[TRS 78]
TRS80 : ***LEVEL II,*** Radio Shack, Ft. Worth, Texas 76102

[WADLER 76]
WADLER P. L. : ***Analysis of an Algorithm for Real Time Garbage Collection,*** C.A.C.M., Vol. 19 No. 9 pp. 491-500, September 1976.

[WAITE 73]
WAITE W. M. : ***Implementing Software for Non-numeric Applications,*** Prentice-Hall Inc., Englewod Cliffs, N. J., 1973.

[WEINREB 79]
WEINREB D. & MOON D. A. : ***LISP MACHINE MANUAL,*** M.I.T. Cambridge, Mass., January 1979.

[WEISSMAN 67]
WEISSMAN C. : ***LISP 1.5 Primer,*** Dickenson Publishing Company Inc., Belmont, California, 1967.

[WEIZENBAUM 68]
WEIZENBAUM J. : *The FUNARG Problem Explained,* M.I.T. Project MAC, March 1968.

[WERTZ 74]
WERTZ H. : LISP CAB500, *Rapport de Recherche goupe II équipe 9,* Université de Paris 8 - Vincennes, 1974-1975.

[WERTZ 77]
WERTZ H. : *VLISP - AID,* Université de Paris 8 - Vincennes, RT 10-77, Juillet 1977.

[WERTZ 78]
WERTZ H. : *Un système de compréhension, d'amélioration et de correction de programmes incorrects,* Thèse de 3ème cycle, Université Paris VI, Juillet 1978.

[WERTZ 79]
WERTZ H. : *Computer Aided Education for Mentaly Retarded Children,* Proc. International Symposium on Computer and Education, Dusseldorf, RFA, Mars 1979.

[WHITE 76]
WHILE J. L. : *Mail from JONL at MIT-ML to LISP-discussion at MIT-AI,* February 1976.

[WHITE 78]
WHITE J. L. : *Programm is Data,* History of Programming Languages Conference, ACM SIGPLAN Notices, Vol. 13 No. 8 pp. 217-223, 1978.

[WINSTON 77]
WINSTON P. H. : *Artificial Intelligence,* Addison-Wesley Company Inc., 1977.

[XGPFONT 74]
XGP FONT CATALOG, M.I.T. Artificial Intelligence Laboratory, Working Paper 72, May 1974.

[ZILOG 78]
Z80 : CPU Technical Manual, Zilog Inc., 1979.

INDEX BIBLIOGRAPHIQUE

APPENDICE A

RESUME DE LA MACHINE VCMC2

Notations utilisées pour décrire les instructions :

<s>	l'opérande source
<d>	l'opérande destination
→	est transféré dans
↔	est échangé avec
CC=	la nouvelle valeur du code condition.
CVAL	l'attribut CVAL d'un symbole atomique
PLIST	l'attribut PLIST d'un symbole atomique
FVAL	l'attribut FVAL d'un symbole atomique
FTYP	l'attribut FTYP d'un symbole atomique
PTYP	l'attribut PTYP d'un symbole atomique
+	l'addition
-	la soustraction
*	la multiplication
/	la division
\	le reste de la division
ou	le ou inclusif logique
et	le et logique
oux	le ou exclusif logique
CAR	la partie CAR d'un doublet
CDR	la partie CDR d'un doublet
(x . y)	un doublet de liste, x est la partie CAR y est la partie CDR
SP	le registre pointeur de pile
(SP)	le contenu du sommet de pile
IX	le registre d'index
x[y]	un opérande indéxé i.e. un opérande dont l'adresse est égale à : x + y

Opérande | NIL

Opérandes | A1, A2, A3 ou A4

Opérande | TST

Opérande | 'objet-VLISP ou (adresse-mémoire)

Opérande | (@ adresse-memoire)

Continuation | [NOP] ou bien omission du champ

Continuation | [JUMP (adresse-memoire)]

Continuation | [CALL (adresse-mémoire)])

Continuation | [RETURN]

MOVE	<s> → <d> & CC= T

ADD	<s> + <d> → <d> & CC= T
SUB	<s> - <d> → <d> & CC= T
MUL	<s> * <d> → <d> & CC= T
QUO	<s> / <d> → <d> & CC= T
REM	<s> \ <d> → <d> & CC= T
LOGOR	<s> ou <d> → <d> & CC= T
LOGAND	<s> et <d> → <d> & CC= T
LOGXOR	<s> oux <d> → <d> & CC= T

CVAL	(CVAL <s>) → <d> & CC= T
SCVAL	<s> → (CVAL <d>) & CC= T
PLIST	(PLIST <s>) → <d> & CC= T
SPLIST	<s> → (PLIST <d>) & CC= T
FVAL	(FVAL <s>) → <d> & CC= T
SFVAL	<s> → (FVAL <d>) & CC= T
FTYP	(FTYP <s>) → <d> & CC= T
SFTYP	<s> → (FTYP <d>) & CC= T
PTYP	(PTYP <s>) → <d> & CC= T
SPTYP	<s> → (PTYP <d>) & CC= T

ADD	<s> + <d> → <d> & CC= T
SUB	<s> - <d> → <d> & CC= T
MUL	<s> * <d> → <d> & CC= T
QUO	<s> / <d> → <d> & CC= T
REM	<s> \ <d> → <d> & CC= T
LOGOR	<s> ou <d> → <d> & CC= T
LOGAND	<s> et <d> → <d> & CC= T
LOGXOR	<s> oux <d> → <d> & CC= T

CAR	(CAR <s>) → <d> & CC= T
CDR	(CDR <s>) → <d> & CC= T
SCAR	<s> → (CAR <d>) & CC= T
SCDR	<s> → (CDR <d>) & CC= T
CONS	(<s> . <d>) → <d> & CC= T
XCONS	(<d> . <s>) → <d> & CC= T

STACK	SP → <d> & CC= T
SSTACK	<s> → SP & CC= T
TOPST	(SP) → <d> & CC= T
XTOPST	(SP) ↔ <s> & CC= T

INDEX	IX → <d>	& CC= T
SINDEX	<s> → IX	& CC= T
MOVEX	<s> → <d>[IX]	& CC= T
XMOVE	<s>[IX] → <d>	& CC= T

JUMPX	<s>[<d>] → PC	& CC= NIL
DISPT	si (LITATOM <d>) <s>[0] → PC	& CC= NIL
	si (NUMBP <d>) <s>[1] → PC	& CC= NIL
	si (LISTP <d>) <s>[2] → PC	& CC= NIL

TATOM	CC= (LITATOM <s>)
FATOM	CC= (NOT (LITATOM <s>))
TNUMB	CC= (NUMBP <s>)
FNUMB	CC= (NOT (NUMPB <s>))
TLIST	CC= (LISTP <s>)
FLIST	CC= (NOT (LISTP <s>))

EQ	CC= (<s> = <d>)
NEQ	CC= (<s> ≠ <d>)

GE		CC= (GE <s> <d>)
GT		CC= (GT <s> <d>)
LT		CC= (LT <s> <d>)
LE		CC= (LE <s> <d>)
SUBTZ	<d> - <s> → <d>	& CC= (ZEROP <d>)
SUBFZ	<d> - <s> → <d>	& CC= (NOT (ZEROP <d>))

NOP		CC= T
STOP		
PRSTACK	<s>	CC= T
PRSTAT		CC= T

READ	entrée → <d>	CC= T
PRINI	<s> → sortie	CC= T
TERPRI	saut de ligne	CC= T

IN	flux d'entrée → <d>	

INTERN	forme interne de <s> → <d>	

OUT	<s> → flux de sortie	

PLEN	(PLENGTH <s>)	→ <d>
PNAM	(PNAM <s>)	→ <d>

PUSH	<s>	équivalent à	MOVE	<s>,TST
POP	<d>	équivalent à	MOVE	TST,<d>
TNIL	<s>	équivalent à	EQ	<s>,NIL
FNIL	<s>	équivalent à	NEQ	<s>,NIL

TRACE	positionne le mode trace
UNTRACE	enlève le mode trace
STEP	positionne le mode pas-à-pas
UNSTEP	enlève le mode pas-à-pas
SILENCE	sauve l'état courant des modes et passe en mode non-trace
REVIVE	repasse dans les modes sauvés par le dernier SILENCE

DATA	objet-VLISP ou (adresse-mémoire)
BLOCK	nombre , objet-VLISP ou (adresse-mémoire)

COMMENT	
ENTRY	<nom>,<FTYP>,<PTYP>

APPENDICE B

```
;                 V C M C 2 M   .   V L I                     ;
;                                                             ;
;             Simulateur de la machine VCMC2                  ;
;-------------------------------------------------------------;
;           Ce fichier doit être édité au moyen de :          ;
;              (CROSSF J2M () T T {NIL/T} T T)                ;
;                   {NIL/T} si VERSATEC                       ;
;-------------------------------------------------------------;
;                                                             ;
;                     Jérôme CHAILLOUX                        ;
;                                                             ;
;           Département d'Informatique                        ;
;           Université de Paris 8 - Vincennes                 ;
;           Route de la Tourelle 75571 Paris Cédex 12         ;
;           Tél : 374 12 50 poste 299                         ;
;                                                             ;
;           L.I.T.P.    (CNRS LA 248)                         ;
;           2 Place Jussieu 75221 Paris Cédex 05              ;
;           Tél : 336 25 25 poste 53-70                       ;
;                                                             ;
;           I.R.C.A.M.                                        ;
;           31 Rue St Merri 75004 Paris                       ;
;           Tél : 277 12 33 poste 48-48                       ;
;-------------------------------------------------------------;
;                                                             ;
; règles de reconnaissance des identificateurs :              ;
;                                                             ;
;   1er car.         signification                            ;
;                                                             ;
;      &      fonctions d'échappements (ESCAPEs)              ;
;      ~      variables globales à tout le simulateur         ;
;      #      variables libres pour certaines fonctions       ;
;             (mais liées par des fnts du simulateur)         ;
;      /      indicateurs sur P-listes                        ;
;      ?      indicateurs du simulateur  (e.g. T ou NIL)      ;
;                                                             ;
;-------------------------------------------------------------;
```

```
; Prologue standard ;
; ___________________ ;
; silence ! ;

(STATUS 2 1 2)

;;
; Pour pouvoir lire ce fichier en tous temps ... ;
;;

(PROGN (SETQ READ.TABLE (READ.TABLE)) (READ.STD) NIL)

; Pour prettyprinter correctement le code ;

(DF CODE (laux) laux)


; Définition du format (TTAB x) = →x ;

(DE → (nbaux) ['→ nbaux])

(DMO → (nbaux) (TTAB nbaux))

(DMC → () ['→ (READ)])

; Définition du format (TERPRI x) = |x ;

(DE | (nbaux) ['| nbaux])

(DMO | (nbaux) (TERPRI nbaux))

(DMC | () ['| (READ)])
```

```
; Fonctions utiles : SORTL PRINTL PRININST ;
; ___________________________________________ ;

(DE SORTL (L ;; RESL N Q L2 LL)
    ; trie la liste d'atomes L , une version de QUICKSORT ;
    ; Patrick GREUSSAY. Août 78 ;
    (IF (NULL L)
        NIL
        (SETQ RESL [NIL])
        (PUSH (LENGTH L) L)
        (LET (L1)
            ; L1 DOIT être locale, voyez-vous pourquoi ? ;
            (SETQ L1 (POP) N (POP))
            (IF (EQN N 1) (LESCAPE L1))
            (SETQ Q (LOGSHIFT N -1))
            (PUSH Q L1)
            (SETQ L1 (NTH Q L1))
            (PUSH (PLUS (LOGAND N 1) Q) (CDR L1))
            (RPLACD L1 NIL)
            (SETQ L1 (SELF) L2 (SELF) LL (RPLACD RESL L1))
            (WHILE L1
                (IF (SORT (CAR L2) (CAR L1))
                    (SETQ N L1 L1 L2 L2 N N (RPLACD LL L1)))
                (SETQ LL L1 L1 (CDR L1)))
            (RPLACD LL L2)
            (CDR RESL))))

(DE PRINTL (s n)
    ; édite l'objet 's' sous forme packée ;
    ; n est le niveau d'édition ;
    (IF (ATOM s)
        (PRIN1 s)
        (IF (GE n 4)
            (PRIN1 '/...)
            (PRIN1 (n '(/( { <))
            (IF s (PRINTL (NEXTL s) (ADD1 n)))
            (IF s (PRINTL (NEXTL s) (ADD1 n)))
            (IF s (PRINTL (NEXTL s) (ADD1 n)))
            (IF s (PRINTL (NEXTL s) (ADD1 n)))
            (AND s (PRIN1 '/...))
            (PRIN1 (n '(/) } >))))))

(DE PRININST (i)
    ; imprime l'instruction i ;
    (IF (ATOM i) (LESCAPE (PRIN1 i) (PRINC ':) (TERPRI)))
    (PRIN1 →5 (NEXTL i) →14 (NEXTL i))
    (PRINC ',)
    (PRIN1 (NEXTL i))
    (IFN i NIL (PRINC ',) (PRIN1 ['LIST . i]))
    (TERPRI))
```

```
; Variables globales du simulateur ;
; ———————————————————————————————— ;
;;
; ~interpreter : liste code de l'interprète VCMC2 ;
(OR (BOUNDP '~interpreter) (SETQ ~interpreter NIL))
; ~pprinter : liste code de l'imprimeur VCMC2 ;
(OR (BOUNDP '~pprinter) (SETQ ~pprinter NIL))
; ~reader : liste code du lecteur VCMC2 ;
(OR (BOUNDP '~reader) (SETQ ~reader NIL))
; ~lregn : liste des noms des registres normaux ;
(SETQ ~lregn '(A1 A2 A3 A4))
; ~lformat : liste des formats d'instruction ;
(SETQ ~lformat '(1MOT 2MOTS 3MOTS 4MOTS))
; ~loper : liste des opérandes d'une instruction ;
(SETQ ~loper '(null A1 A2 A3 A4 TST immediat direct))
; ~lcont : liste des continuations d'une instruction ;
(SETQ ~lcont '(NOP JUMP CALL RETURN))
; ~linstr : liste des intructions légales de VCMC2 ;
(SETQ
   ~linstr
    (SORTL
     '(MOVE CAR CDR SCAR SCDR INDEX SINDEX MOVEX XMOVE CVAL SCVAL PLIST
       SPLIST FTYP SFTYP FVAL SFVAL PTYP SPTYP CONS XCONS STACK SSTACK
       TOPST XTOPST ADD SUB MUL DIV REM LOGOR LOGAND LOGXOR NOP JUMPX
       DISPT TATOM FATOM TLIST FLIST TNUMB FNUMB EQ NEQ LT LE GT GE
       SUBTZ SUBFZ READ IN INTERN PRINI TERPRI OUT PLEN PNAM PRSTACK
       STOP)))
; ~ltypsubr : liste des types des SUBR ;
(SETQ ~ltypsubr '(0SUBR 1SUBR 2SUBR 3SUBR NSUBR FSUBR))
; ~stacklength : taille max de la pile ;
(SETQ ~stacklength 4000)
```

```
; Indicateurs de P-listes globaux utilisés dans le simulateur ;
; ____________________________________________________________ ;
;
                Attributs atomiques : sous les variables simulées

                        /cval
                        /plist
                        /fval
                        /ftyp
                        /ptyp

                Compteurs du simulateur :

                        /inst           occurrence d'une intruction
                        /operand        occurrence d'un opérande
                        /format         occurrence d'un format
                        /contin         occurrence d'une continuation
                        /label          passage à une étiquette
                        /typsubr        type des SUBR

  ;
;;
; Indicateurs de contrôle du simulateur ;
; _____________________________________ ;
;;
; trace de la fonction POINT ;

(OR (BOUNDP '?pointrace) (SETQ ?pointrace NIL))

; trace du contenu des registres ;

(OR (BOUNDP '?contrace) (SETQ ?contrace NIL))

; trace des instructions exécutées ;

(OR (BOUNDP '?instrace) (SETQ ?instrace NIL))

; trace du contenu de la pile et du registre d'index ;

(OR (BOUNDP '?prinstack) (SETQ ?prinstack NIL))

; trace en mode stepper ;

(OR (BOUNDP '?stepper) (SETQ ?stepper NIL))
```

```
; Lancement de la machine : VCMC2 ;
; ———————————————————————————————— ;

(DE VCMC2
 (LCOD ; liste des intructions ;
  ?stat ; =T si on imprime des statistiques en fin ;
  ?instat ; =T si on compte les occurrences des instructions ;
  ?labstat ; =T si on compte les passages aux étiquettes ;
  larg ; liste-état des 4 lers registres ;
  ;;
  A1 ; ler reg. ;
  A2 ; 2ème reg ;
  A3 ; 3ème reg ;
  A4 ; 4ème reg ;
  PC ; compteur ordinal courant ;
  ; liste des instructions restant à faire ;
  ST ; contenu courant de la pile ;
  INDEX ; contenu courant du registre d'index ;
  ;;
  #ninst ; nb d'instructions exécutées ;
  #nincr ; nb d'instructions en incremental ;
  #ncont ; nb de continuations ;
  #noper ; nb d'opérandes évalués ;
  #nword ; nb de mots instructions lus ;
  #nbcons ; nb de CONS effectués ;
  #stackl ; taille max de la pile ;
  runtime ; durée d'exécution de VCMC2 ;
  x ; auxilliaire ;)
   ;;
   ; RESET de la machine VCMC2 ;
   ;;
   (SETQ
      A1 (CAR larg) ; init ler arg ;
      A2 (CADR larg) ; init 2ème arg ;
      A3 (CADDR larg) ; init 3ème arg ;
      A4 larg ; init liste des arguments ;
      ST '(((STOP)) *eos*) ; contenu courant de la pile ;
      PC LCOD ; liste des instructions ;
      #ninst 0
      #nincr 0
      #noper 0
      #nword 0
      #nbcons 0
      #stackl 2
      runtime (RUNTIME))
   ;;
   ; init des compteurs des types d'instruction ;
   ;;
   (IF ?instat (MAPC ~linstr (LAMBDA (L) (PUT L 0 '!inst))))
   (IF ?instat (MAPC ~lformat (LAMBDA (L) (PUT L 0 '!format))))
   (IF ?instat (MAPC ~lcont (LAMBDA (L) (PUT L '0 '!contin))))
   (IF ?instat (MAPC ~loper (LAMBDA (L) (PUT L '0 '!operand))))
```

```
; Séquenceur principal ;
;;
(LET
 ((inst) (R1) (R2) (continuation) (codecondition) (xaux) (valregs))
   ; Il faut pas mal de variables locales à XCT ;
   (ESCAPE &STOP
      (WHILE (LISTP PC)
         (IF ?instrace (PRININST (CAR PC)))
         (IFN ?stepper
            NIL
            (UNTIL (EQ (PROGN (TYO (CASCII '↑)) (TYI)) 32)
               (IF (EQ (TYS) 10) (TYI))
               (PRINT (EVAL (READ)))))
         (IF (LISTP (CAR PC))
            ; C'est une instruction ;
            (XCT (NEXTL PC))
            ; C'est une étiquette ;
            (IF (OR (NULL ?labstat) (NULL (CAR PC)))
               (NEXTL PC)
               (SETQ x (GET (CAR PC) '!label))
               (PUT (CAR PC) (IF x (ADD1 x) 1) '!label)
               (NEXTL PC))))
      (PRINT "Machine VCMC2 - HALT  sur : " PC)))
```

```
; Impression des statistiques ;
;;
(IF ?stat
   (PROGN
      (PRINT "Nb d'instructions exécutées :" #ninst)
      (PRINT "Temps d'une instruction     :"
       (// (- (RUNTIME) runtime) (FLOAT #ninst)) " ms.")
      (PRINT "Nb de CONS effectués        :" #nbcons)
      (PRINT "Taille maximum de la pile   :" (DIFFER #stackl 2))))
;;
; Impression des comptages dynamiques ;
;;
(IF ?instat
   (TABSTAT "Occurrences dynamiques des instructions" ~linstr '!inst
    #ninst 0 [" Nb d'instructions exécutées : " #ninst]))
(IF ?instat
   (TABSTAT "Occurrences dynamiques des formats" ~lformat '!format
    #ninst 0
    [" Nb d'instructions exécutées : "
     #ninst
     " Nb de mots des instructions : "
     #nword]))
(IF ?instat
   (TABSTAT "Occurrences dynamiques des continuations" ~lcont
    '!contin #ncont #ninst
    [" Nb de continuations         : "
     #ncont
     " Nb d'instructions exécutées : "
     #ninst]))
(IF ?instat
   (TABSTAT "Occurrences dynamiques des opérandes" ~loper '!operand
    #noper #ninst
    [" Nb d'opérandes évalués      : "
     #noper
     " Nb d'instructions exécutées : "
     #ninst]))
;;
; Impression des passages aux étiquettes ;
;;
(IF ?labstat
   (PROGN
      (SETQ x (GET 'EVALA1 '!label))
      (TABSTAT "Passage aux étiquettes"
       (SORTL (MAPCT (OBLIST) (LAMBDA (x) (IF (GET x '!label) x))))
       '!label x #ninst
       [" Nb de passages dans EVALA1  : "
        x
        " Nb d'instructions exécutées : "
        #ninst])))
; Ca ramène toujours A1 ;
A1)
```

```
; Décodage des opérandes : VALUESOURCE  POINT ;
; ————————————————————————————————————————— ;

(DM VALUESOURCE (l)
   ; macro-génère (CAR (POINT x NIL)) ;
   (RPLACB l ['CAR ['POINT (CADR l)]]))

(DE POINT (adef idest ;; xaux)
   ; ramène le pointeur 'adef' (adresse effective d'opérande) ;
   ; idest = T si opérande destination ;
   ; TRACE si ?pointrace = T ;
   (IF ?instat (INCR #noper))
   (SETQ
      POINT
       (COND
          ((MEMQ adef '(A1 A2 A3 A4))
             (IF ?instat (PUTINCR adef '!operand))
             adef)
          ((NULL adef)
             (IF ?instat (PUTINCR 'null '!operand))
             (IFN idest
                [NIL]
                (MACHERR 'POINT "NIL comme opérande destination.")))
          ((EQ adef 'TST)
             (IF ?instat (PUTINCR adef '!operand))
             (IF (ATOM ST) (MACHERR 'POINT "Pile détruite"))
             (IF idest
                ; cas destination : PUSH ;
                (PROGN
                   (SETQ xaux (LENGTH ST))
                   (AND
                      (GT xaux #stackl)
                      (SETQ #stackl xaux)
                      (IF (GE xaux ~stacklength)
                         (MACHERR 'POINT "Pile pleine.")))
                   (SETQ TST [NIL] ST [TST . ST])
                   ST)
                ; cas source : POP ;
                (IF (EQ (CAR ST) '*eos*)
                   (MACHERR 'POINT "Pile vide.")
                   (PROG1 ST (SETQ TST (NEXTL ST))))))
          ((ATOM adef) (MACHERR 'POINT "Opérande inconnu :" adef))
          (T ; adresse ( ) donc par PC auto-incrément ;
            (IF ?instat
               (IF (EQ (CAR adef) '@)
                  (PUTINCR 'direct '!operand)
                  (PUTINCR 'immediat '!operand)))
            (COND
               ((EQ (CAR adef) 'DATA) (SELF (CADR adef)))
               ((AND (CDR adef) (EQ (CAR adef) QUOTE)) (CDR adef))
               ((EQ (CAR adef) '@)
                  (SETQ
                     xaux
                      (OR
                         (MEMQ (CADR adef) ~reader)
                         (MEMQ (CADR adef) ~pprinter)
                         (MEMQ (CADR adef) ~interpreter)))
                  (IF (NEQ (CAADR xaux) 'DATA)
                     (MACHERR 'POINT "@ sans DATA.")
                     (CDADR xaux)))
               ((EQ (CAR adef) '*VAL*)
                  (SELF (EPROGN (CDR adef)) idest))
```

```
                    ((GET (CAR adef) '!point))
                    ((SETQ
                        xaux
                         (OR
                            (MEMQ (CAR adef) ~reader)
                            (MEMQ (CAR adef) ~pprinter)
                            (MEMQ (CAR adef) ~interpreter)))
                       (IFN xaux
                          (MACHERR 'POINT "Etiquette inconnue.")
                          (SETQ xaux [xaux])
                          (PUT (CAR adef) xaux '!point)
                          xaux))
                    (T (MACHERR 'POINT "Opérande inconnu :" adef)))))
      (IF ?pointrace (PRINT →15 'POINT adef '= (CAR POINT)))
      POINT)
```

```
; Fnts spéciales de la micro-machine : MACHERR MOVE JUMP PRSTACK ;
; ——————————————————————————————————————————————————————————————— ;

(DE MACHERR (V L . xaux)
   ; erreur de la machine VCMC2 ;
   ; 1) imprime les messages d'erreur ;
   (PRINT "***** Erreur machine VCMC2 dans :" V)
   (PRIN1 →6 L)
   (IF xaux (PRINT xaux) (TERPRI))
   (PRINT →6 "Instruction : " linst)
   ; 2) réalise l'interruption ;
   (XCT '(NOP [CALL (MERROR)]))
   ; 3) retourne NIL ;
   NIL)

(DE MOVE (R1 R2)
   ; réalise le transfert de R1 vers R2 ;
   (SETQ R1 (VALUESOURCE R1))
   (SET (POINT R2 T) R1))

(DE JUMP (R1 ;; ?instat)
   ; ?instat est lié pour ne pas compter les opérandes des cont. ;
   (SETQ R1 (VALUESOURCE R1))
   (IF (OR (NULL R1) (LISTP R1))
      ; C'est une étiquette VCMC2 ;
      (SETQ PC R1)
      ; C'est une étiquette  inconnue ;
      (MACHERR 'JUMP "Adresse inconnue : " R1)))

(DE PRSTACK (n stack)
   ; imprime les n premiers contenus de la pile stack ;
   ; réalise donc l'instruction VCMC2 PRSTACK ;
   (IF (NULL stack) (LESCAPE))
   (TTAB 12)
   (IF (ZEROP n)
      (LESCAPE)
      (LET ((s (CAR stack)) (d 2))
         (IF (ATOM s)
            (PRIN1 s)
            (IF (ZEROP d)
               (PRIN1 '&)
               (IF (GE (LENGTH s) 100)
                  (PRIN1 (CAR s) ':)
                  (PRIN1 '/()
                  (SELF (CAR s) (SUB1 d))
                  (IF (NULL (CDR s))
                     (PRIN1 '/))
                     (SELF (CADR s) (SUB1 d))
                     (IF (NULL (CDDR s)) (PRIN1 '/)) (PRIN1 '/.../))))))))
      (TERPRI)
      (SELF (SUB1 n) (CDR stack))))
```

```
; Exécuteur des instructions : XCT ;
; ------------------------------------------ ;

(DE XCT (linst)
   ; exécute 1 instruction VCMC2 : 'linst' ;
   (IF ?stat
      (PROGN
         (INCR #ninst)
         (INCR #nincr)
         (IF (AND (ZEROP (REM #ninst 1000)) (IRCAMP) (LZP (TRMOP 547)))
            ; Je suis à l'IRCAM sur DATA-MEDIA ;
            (DISPLAY
             (APPEND [127 14 127 12 121 98]
              (MAPCAR (EXPLODE #ninst) 'CASCII))))))
   (IF ?contrace (SETQ valregs [A1 A2 A3 A4]))
   ;;
   ; Fetch instruction ;
   ;;
   (IF (ATOM linst) (LESCAPE (MACHERR 'XCT "Exécution d'un atome.")))
   ;;
   ; décodage opérandes ;
   ;;
   (SETQ
      l linst
      inst (NEXTL l) ; Code instruction ;
      R1 (NEXTL l) ; 1er opérande ;
      R2 (NEXTL l) ; 2ème opérande ;
      continuation l ; champ continuation ;
      codecondition T ; CODE condition T  à priori ;)
   (IFN (LITATOM inst) (MACHERR 'XCT "Instruction incorrecte."))
   ;;
   ; Comptage du type de l'instruction  et du format ;
   ;;
   (IF ?instat (PUTINCR inst '!inst))
   (IF ?instat
      (PROGN
         (SETQ x 1)
         (IF (LISTP R1) (INCR x))
         (IF (LISTP R2) (INCR x))
         (IF
          (AND
             (LISTP continuation)
             (MEMQ (CAR continuation) '(JUMP CALL))) (INCR x))
         (SETQ #nword (PLUS #nword x))
         (PUTINCR
          (CASSOC x '((1 . 1MOT) (2 . 2MOTS) (3 . 3MOTS) (4 . 4MOTS)))
          '!format)))

```

```
        ; Exécution ;
        ;;
        (SELECTQ  inst
           ;;
           ; Les Pseudo-instructions ;
           ;;
           (ENTRY
              (PUT R1 (OR continuation R1) '!fval)
              (PUT R1
                 (CASSQ R2
                   '((0SUBR . 1) (1SUBR . 2) (2SUBR . 3) (3SUBR . 4)
                     (NSUBR . 5) (FSUBR . 6)))
                 '!ftyp))
           ((DATA BLOCK) ; provoque irrémédiablement une erreur ;
              (MACHERR 'XCT "Exécution d'une DATA."))
           (TRACE ; lance une TRACE ;
              (SETQ ?instrace T ?contrace T ?prinstack T))
           (UNTRACE ; enlève la TRACE ;
              (SETQ ?instrace NIL ?contrace NIL ?prinstack NIL))
           (STEP ; lance un STEP ;
              (SETQ ?instrace T ?contrace T ?prinstack T ?stepper T))
           (UNSTEP ; enlève le STEP ;
              (SETQ ?instrace NIL ?contrace NIL ?prinstack NIL ?stepper NIL))
           (SILENCE ; inhibe toutes les traces ;
              (PUSH ?instrace ?contrace ?prinstack ?pointrace ?stepper)
              (SETQ
                 ?instrace ()
                 ?contrace ()
                 ?prinstack ()
                 ?pointrace ()
                 ?stepper ()))
           (REVIVE ; restaure les traces inhibées par SILENCE ;
              (SETQ
                 ?stepper (POP)
                 ?pointrace (POP)
                 ?prinstack (POP)
                 ?contrace (POP)
                 ?instrace (POP)))
           (;; (SETQ continuation NIL))
           (PRSTAT ; imprime les statistiques en incrémental ;
              (IFN ?stat
                 NIL
                 (PRINT "Nb d'instructions exécutées :" #nincr)
                 (PRINT "Nb de CONS réalisés         :" #nbcons)
                 (PRINT "Taille maximum de la pile   :" #stackl)
                 (SETQ #nincr 0 #nbcons 0 #stackl 0)))
           ;;
           ; Instructions de transferts de base ;
           ;;
           (MOVE (MOVE R1 R2))
           (CAR (SETQ R1 (VALUESOURCE R1)) (SET (POINT R2 T) (CAR R1)))
           (CDR (SETQ R1 (VALUESOURCE R1)) (SET (POINT R2 T) (CDR R1)))
           (SCAR (SETQ R1 (VALUESOURCE R1)) (RPLACA (CAR (POINT R2 T)) R1))
           (SCDR (SETQ R1 (VALUESOURCE R1)) (RPLACD (CAR (POINT R2 T)) R1))
           ;;
           ; Instructions manipulant l'index ;
           ;;
           (INDEX ; lecture de la valeur de l'index ;
              (SET (POINT R1 T) INDEX))
           (SINDEX ; écriture de l'index ; (SETQ INDEX (VALUESOURCE R1)))
           (MOVEX ; opérande normal → opérande indexé ;
              (SETQ R1 (VALUESOURCE R1))
              (SETQ R2 (CNTH (PLUS (VALUESOURCE R2) 2) INDEX))
```

```
        (IF (NEQ (CAR R2) 'DATA)
           (MACHERR 'XCT "MOVEX : sans DATA.")
           (SET (CDR R2) R1)))
    (XMOVE ; opérande indexé → opérande normal ;
        (SETQ R1 (CNTH (PLUS (VALUESOURCE R1) 2) INDEX))
        (IF (NEQ (CAR R1) 'DATA)
           (MACHERR 'XCT "XMOVE : sans DATA.")
           (SET (POINT R2 T) (CADR R1))))
    ;;
    ; Fonctions spéciales sur les atomes ;
    ; Tous les attributs sont sur les PLIST ;
    ;;
    (CVAL ; les CVAL sont sur la PLIST pour éviter les ;
        ; collisions avec les variables du simulateur ;
        (SETQ R1 (VALUESOURCE R1))
        (SETQ xaux (GETL R1 '(!cval)))
        (SET (POINT R2 T)
         (IF (LISTP xaux)
            (CADR xaux)
            (IF (MEMQ R1 '(NIL T LAMBDA)) R1 '~UNDEF))))
    (SCVAL
        (SETQ R1 (VALUESOURCE R1))
        (PUT (CAR (POINT R2 T)) R1 '!cval))
    (PLIST
        (SETQ R1 (VALUESOURCE R1))
        (SET (POINT R2 T) (GET R1 '!plist)))
    (SPLIST
        (SETQ R1 (VALUESOURCE R1))
        (PUT (CAR (POINT R2 T)) R1 '!plist))
    (FVAL
        (SETQ R1 (VALUESOURCE R1))
        (SET (POINT R2 T) (GET R1 '!fval)))
    (SFVAL
        (SETQ R1 (VALUESOURCE R1))
        (PUT (CAR (POINT R2 T)) R1 '!fval))
    (FTYP
        (SETQ R1 (VALUESOURCE R1))
        (SET (POINT R2 T) (GET R1 '!ftyp)))
    (SFTYP
        (SETQ R1 (VALUESOURCE R1))
        (PUT (CAR (POINT R2 T)) R1 '!ftyp))
    (PTYP
        (SETQ R1 (VALUESOURCE R1))
        (SET (POINT R2 T) (GET R1 '!ptyp)))
    (SPTYP
        (SETQ R1 (VALUESOURCE R1))
        (PUT (CAR (POINT R2 T)) R1 '!ptyp))
    ;;
    ; Instructions de création de doublets ;
    ;;
    (CONS
        (INCR #nbcons)
        (SETQ R1 [(VALUESOURCE R1) . (VALUESOURCE R2)])
        (SET (POINT R2 T) R1))
    (XCONS
        (INCR #nbcons)
        (SETQ R1 (XCONS (VALUESOURCE R1) (VALUESOURCE R2)))
        (SET (POINT R2 T) R1))
    ;;
    ; Instructions sur la pile ;
    ;;
    (STACK ; lecture du pointeur de pile ; (SET (POINT R1 T) ST))
    (SSTACK ; écriture du pointeur de pile ;
```

```
        (SETQ ST (VALUESOURCE R1)))
    (TOPST ; lecture du sommet de la pile ;
        (SET (POINT R1 T) (CAR ST)))
    (XTOPST ; échange du sommet de la pile ;
        (SETQ xaux (CAR ST))
        (SET ST (VALUESOURCE R1))
        (SET (POINT R1 T) xaux))
    ;;
    ; Instructions arithmétiques ;
    ;;
    (ADD
        (SETQ R1 (PLUS (VALUESOURCE R1) (VALUESOURCE R2)))
        (SET (POINT R2 T) R1))
    (SUB
        (SETQ R1 (VALUESOURCE R1))
        (SETQ R1 (DIFFER (VALUESOURCE R2) R1))
        (SET (POINT R2 T) R1))
    (MUL
        (SETQ R1 (TIMES (VALUESOURCE R1) (VALUESOURCE R2)))
        (SET (POINT R2 T) R1))
    (DIV
        (SETQ R1 (VALUESOURCE R1))
        (SETQ R1 (QUO (VALUESOURCE R2) R1))
        (SET (POINT R2 T) R1))
    (REM
        (SETQ R1 (VALUESOURCE R1))
        (SETQ R1 (REM (VALUESOURCE R2) R1))
        (SET (POINT R2 T) R1))
    ;;
    ; Instructions logiques ;
    ;;
    (LOGOR
        (SETQ R1 (LOGOR (VALUESOURCE R1) (VALUESOURCE R2)))
        (SET (POINT R2 T) R1))
    (LOGAND
        (SETQ R1 (LOGAND (VALUESOURCE R1) (VALUESOURCE R2)))
        (SET (POINT R2 T) R1))
    (LOGXOR
        (SETQ R1 (LOGXOR (VALUESOURCE R1) (VALUESOURCE R2)))
        (SET (POINT R2 T) R1))
    ;;
    ; Instructions de controle (elles sont réalisées ;
    ; en utilisant le champ continuation) ;
    ;;
    (NOP ; pour réaliser les instructions inconditionnelles ;)
    ;;
    ; Instructions d'aiguillage ;
    ;;
    (JUMPX
        (SETQ R1 (VALUESOURCE R1))
        (JUMP (CNTH (PLUS (VALUESOURCE R2) 2) R1)))
    (DISPT
        (SETQ R1 (VALUESOURCE R1) R2 (VALUESOURCE R2))
        (JUMP (CNTH (COND
                  ((LITATOM R2) 2)
                  ((NUMBP R2) 3)
                  (T 4)) R1)))
    ;;
    ; Instruction qui positionnent le codecondition ;
    ;;
    (TATOM (SETQ codecondition (LITATOM (VALUESOURCE R1))))
    (FATOM (SETQ codecondition (NULL (LITATOM (VALUESOURCE R1)))))
    (TLIST (SETQ codecondition (LISTP (VALUESOURCE R1))))
```

```
(FLIST (SETQ codecondition (ATOM (VALUESOURCE R1))))
(TNUMB (SETQ codecondition (NUMBP (VALUESOURCE R1))))
(FNUMB (SETQ codecondition (NULL (NUMBP (VALUESOURCE R1)))))
(EQ (SETQ codecondition (EQ (VALUESOURCE R1) (VALUESOURCE R2))))
(NEQ (SETQ codecondition (NEQ (VALUESOURCE R1) (VALUESOURCE R2))))
(LT (SETQ codecondition (LT (VALUESOURCE R1) (VALUESOURCE R2))))
(LE (SETQ codecondition (LE (VALUESOURCE R1) (VALUESOURCE R2))))
(GT (SETQ codecondition (GT (VALUESOURCE R1) (VALUESOURCE R2))))
(GE (SETQ codecondition (GE (VALUESOURCE R1) (VALUESOURCE R2))))
(SUBTZ
   (SETQ R1 (VALUESOURCE R1))
   (SETQ R1 (DIFFER (VALUESOURCE R2) R1))
   (SET (POINT R2 T) R1)
   (SETQ codecondition (ZEROP R1)))
(SUBFZ
   (SETQ R1 (VALUESOURCE R1))
   (SETQ R1 (DIFFER (VALUESOURCE R2) R1))
   (SET (POINT R2 T) R1)
   (SETQ codecondition (NULL (ZEROP R1))))
;;
; Instructions spéciales d'entrée ;
;;
(READ ; pour l'interprète seul ; (SET (POINT R1 T) (READ)))
(IN ; pour le READ simulé ;
   (SET (POINT R1 T) (CASCII (READCH))))
(INTERN ; fabrique un atome ;
   (SETQ R1 (CDR (VALUESOURCE R1)))
   (SET (POINT R2 T)
    (APPLY 'GENSYM
     (ESCAPE &INTERN
        (LET (l)
           (IF (OR (NEQ (CAAR R1) 'DATA) (ZEROP (CADAR R1)))
               (&INTERN (REVERSE l))
               (SELF (NEWL l (ASCII (CADR (NEXTL R1))))))))))
;;
; Instructions spéciales de sortie ;
;;
(PRINI ; pour l'interprète seul ; (PRIN1 (VALUESOURCE R1)))
(TERPRI ; pour l'interprète seul ; (TERPRI))
(OUT ; pour l'imprimeur simulé ;
   (SETQ R1 (VALUESOURCE R1))
   (PRINC
    (COND
       ((STRINGP R1) (CAR (MAKLIST R1)))
       ((NUMBP R1) (ASCII R1))
       (T R1))))
(PLEN ; retourne la longeur du P-name ;
   (SETQ R1 (VALUESOURCE R1))
   (SET (POINT R2 T) (IF (STRINGP R1) (STRINGL R1) (PLENGTH R1))))
(PNAM ; retourne la liste des caracteres de <s> dans <d> ;
   (SETQ
      R1 (VALUESOURCE R1)
      R1 (IF (STRINGP R1) (MAKLIST R1) (EXPLODE R1))
      R2 (CDR (VALUESOURCE R2)))
   (MAPC R1
      (LAMBDA (C)
         (RPLACA (CDAR R2) (CASCII C))
         (SETQ R2 (CDR R2)))))
((PRATOM OUTCH)
   (PUSH (STATUS 0))
   ; pas de 1er espace, ni de /, ni de ", ni de préfixe ;
   (STATUS 2 21 24 25 27 28)
   (PRIN1 (VALUESOURCE R1))
```

```
            (STATUS 0 (POP)))
         (OUTBUF (TERPRI))
         ;;
         ; Instructions spéciales ;
         ;;
         (PRSTACK (PRSTACK (VALUESOURCE R1) ST))
         (STOP (&STOP))
         ;;
         ; Instruction illégale ;
         ;;
         ((MACHERR 'XCT "Instruction illégale.")))
      ;;
      ; Traitement du champ continuation ;
      ;;
      (IF (AND ?instat continuation)
         (PROGN (INCR #ncont) (PUTINCR (CAR continuation) '!contin)))
      (IF (AND continuation codecondition)
         (SELECTQ  (CAR continuation)
            (NOP)
            (JUMP (JUMP (CADR continuation)))
            (CALL
               (SETQ ST [PC . ST])
               (SETQ xaux (LENGTH ST))
               (AND (GT xaux #stack!) (SETQ #stack! xaux))
               (JUMP (CADR continuation)))
            (RETURN (JUMP 'TST))
            ((MACHERR 'XCT "Champs continuation illégal."))))
      ;;
      ; Dernières statistiques ;
      ;;
      (IF (AND ?contrace (NOT (MEMQ inst '(SSTACK STACK))))
         (MAPC ~lregn
            (LAMBDA (xaux)
               (OR
                  (EQ (CAR xaux) (CAR valregs))
                  (EQUAL (CAR xaux) (CAR valregs))
                  (PROGN
                     (PRIN1 →10 xaux '=)
                     (PRINTL (CAR xaux) 1)
                     (TERPRI)))
               (NEXTL valregs))))
      (IFN ?prinstack
         NIL
         (STATUS 7 25)
         (TTAB 20)
         (PRIN1 '** 'INDEX '=)
         (PRINTL INDEX 1)
         (TERPRI)
         (TTAB 20)
         (PRIN1 '** 'ST '=)
         (PRINTL ST 1)
         (STATUS 7 0)
         (PRINTLEVEL 1000)
         (TERPRI)))
```

```
; Fonction d'impression d'un tableau : TABSTAT ;
; ————————————————————————————————————————————— ;

(DE TABSTAT
 (titre ; chaîne titre du tableau ;
  clair ; liste des clairs du comptage ;
  indic ; indicateur du comptage ;
  total ; total du comptage ;
  gtot ; grand total ;
  lcomm ; liste des commentaires ;)
   ;;
   (STATUS 2 21 28)
   (PRINT |1 ".B" |1 ".TP " (PLUS (LENGTH lcomm) (LENGTH clair) 7) |1
    "$=$1" ;$<$0; |1
    ".C ; ┌──────────────────────────────────────────────────────┐" |1 ".C ; |"
   →51 "|" |1 ".C ; |" →8 "↑&" titre "\&" →55 "|" |1 ".C ; |" →51 "|")
  (IF lcomm
     (PROGN
        ; Il y a des commentaires ;
        (WHILE lcomm
           (PRINT ".C ; | " (NEXTL lcomm) (NEXTL lcomm) →51 "|"))
        (PRINT ".C ; |" →51 "|")))
  (OR (NUMBP total) (SETQ total 0))
  (OR (NUMBP gtot) (SETQ gtot 0))
  (SETQ total (FLOAT total) gtot (FLOAT gtot))
  (LET (nbaux 0)
     (MAPC clair
        (LAMBDA (laux ;; xaux perc1 perc2)
           (SETQ xaux (OR (NUMBP (GET laux indic)) 0))
           (IF (ZEROP xaux) (LESCAPE))
           (SETQ
              perc1
               (IF (OR (ZEROP total) (ZEROP xaux))
                  0
                  (* (// (FLOAT xaux) total) 100.))
              perc1 (IF (> perc1 0.1) (// (FIX (* perc1 100)) 100.) 0)
              perc2
               (IF (OR (ZEROP gtot) (ZEROP xaux))
                  0
                  (* (// (FLOAT xaux) gtot) 100.))
              perc2 (IF (> perc2 0.1) (// (FIX (* perc2 100)) 100.) 0))
          (PRINT ".C ; |" →8 (INCR nbaux) →13 laux →22 "= " xaux →32
           perc1 →37 " %   " →42 perc2 →47 " %   " →51 "|"))))
  (PRINT ".C ; |" →51 "|" |1
   ".C ; └──────────────────────────────────────────────────────┘" |1 "$>$0"
   ;$<; |1 ".BR")
  (STATUS 1 21 28))
```

```
; Analyse du code : ANACODE  et épilogue ;
; ———————————————————————————————— ;
;;
; Doit être lancée AVANT toute interprétation ;

(DE ANACODE
 (llinst ; est la liste de code à analyser  ;
  ? ; T s'il faut imprimer les stats ;
  ?? ; T s'il faut analyser les instructions  (statiques) ;
  ;;
  lfunct ; liste des fonctions du code ;
  nfunct ; nombre de fonctions du code ;
  ninst ; nombre d'instructions du code ;
  ncont ; nombre de champs continuation du code ;
  ;;
  i ; codop ;
  op1 ; opérande 1 ;
  op2 ; opérande 2 ;
  cont ; continuation  ;)
   ;;
   ; Pose des indicateurs des fonctions standard ;
   ;;
   (SETQ nfunct 0 nword 0 ninst 0 ncont 0)
   (IF ?? (MAPC ~linstr (LAMBDA (L) (PUT L 0 '!inst))))
   (IF ?? (MAPC ~lformat (LAMBDA (L) (PUT L 0 '!format))))
   (IF ?? (MAPC ~lcont (LAMBDA (L) (PUT L '0 '!contin))))
   (IF ?? (MAPC ~loper (LAMBDA (L) (PUT L '0 '!operand))))
   (IF ?? (MAPC ~ltypsubr (LAMBDA (L) (PUT L '0 '!typsubr))))
   ;;
   (PRINT "*** Analyse du code contenu dans  :" llinst)
   ;;
   (MAPC llinst
      (LAMBDA (linst)
         ;;
         ; Analyse ;
         ;;
         (MAP (CDDAR linst)
            (LAMBDA (inst)
               (IF (ATOM (CAR inst)) (LESCAPE))
               (IF
                (OR
                   (NOT (LITATOM (CAAR inst)))
                   (GT (LENGTH (CAR inst)) 5))
                  (LESCAPE (PRINT "CODE altéré..." inst)))
               ; Traitement des MACROS ;
               (COND
                  ((EQ (CAAR inst) 'TNIL)
                     (RPLACB (CAR inst) ['EQ NIL . (CDAR inst)]))
                  ((EQ (CAAR inst) 'FNIL)
                     (RPLACB (CAR inst) ['NEQ NIL . (CDAR inst)]))
                  ((EQ (CAAR inst) 'POP)
                     (RPLACB (CAR inst) ['MOVE 'TST . (CDAR inst)]))
                  ((EQ (CAAR inst) 'PUSH)
                     (RPLACB (CAR inst)
                      ['MOVE (CADAR inst) 'TST . (CDDAR inst)])))
               ; Traitement obligatoire ;
               (COND
                  ((EQ (CAAR inst) 'ENTRY)
                     (INCR nfunct)
                     (IF (MEMQ (CADAR inst) lfunct)
                        NIL
                        (NEWL lfunct (CADAR inst)))
```

```
                 (PUT (CADAR inst)
                    (MEMQ (CADAR inst) (CDDAR linst))
                    '!fval)
                 (PUT (CADAR inst)
                    (CASSQ (CADDAR inst)
                     '((0SUBR . 1) (1SUBR . 2) (2SUBR . 3)
                       (3SUBR . 4) (NSUBR . 5) (FSUBR . 6)))
                    '!ftyp)
                 (PUT (CADAR inst) (CADDDR (CAR inst)) '!ptyp)
                 (IF ?? (PUTINCR (CADDAR inst) '!typsubr)))
              ((EQ (CAAR inst) 'BLOCK) ; expansion des blocks ;
                 (RPLACB inst
                  (NCONC
                   (LET (n (CADR (CAR inst)))
                      (IF (LEZP n)
                         NIL
                         [['DATA (CADDR (CAR inst))]
                          . (SELF (SUB1 n))])) (CDR inst))))
              (T NIL))
          ;;
          ; passe sous format interne ;
          ;;
          (LET (l (CAR inst))
             (SETQ
                codop (NEXTL l)
                op1
                 (IF (AND (MEMQ (CAAR l) '[NCONS]) (CDAR l))
                    NIL
                    (NEXTL l))
                op2
                 (IF (AND (MEMQ (CAAR l) '[NCONS]) (CDAR l))
                    NIL
                    (NEXTL l))
                cont
                 (IF (LISTP (CAR l))
                    ; Format non interné ;
                    (IF (MEMQ (CAAR l) '[NCONS]) (CDAR l) NIL)
                    ; Format déjà interné ;
                    l))
             (RPLACB (CAR inst) [codop op1 op2 . cont]))
          ;;
          ; si ne faut pas de statistiques c'est terminé ;
          ;;
          (IFN ?? (LESCAPE))
          ;;
          ; Analyse statique ;
          ;;
          (IF (MEMQ codop '(BLOCK DATA ENTRY)) (LESCAPE))
          (INCR ninst)
          (INCR nword)
          (PUTINCR codop '!inst)
          ; x est le nombre de mots de l'instruction ;
          (SETQ x 1)
          (MAPC [op1 op2]
             (LAMBDA (op)
                ; pour les 2 opérandes ;
                (COND
                   ((NULL op) (PUTINCR 'null '!operand))
                   ((MEMQ op '(A1 A2 A3 A4 TST))
                      (PUTINCR op '!operand))
                   ((AND (LISTP op) (NEQ (CAR op) '@))
                      (INCR x)
                      (INCR nword)
```

```
                          (PUTINCR 'immediat '!operand))
                       ((AND (LISTP op) (EQ (CAR op) '@))
                          (INCR x)
                          (INCR nword)
                          (PUTINCR 'direct '!operand))
                       (T))))
              (IF cont
                 (PROGN
                    (INCR ncont)
                    (PUTINCR (CAR cont) '!contin)
                    (IF (MEMQ (CAR cont) '(CALL RETURN))
                       (PROGN (INCR x) (INCR nword)))))
              (PUTINCR
               (CASSOC x
                '((1 . 1MOT) (2 . 2MOTS) (3 . 3MOTS) (4 . 4MOTS)))
               '!format)))))
    (PRINT →5 "Nombre de fonctions définies  : " nfunct)
    (IF ??
       (PRINT |1 →5 "|  Nombre d'instructions        :          |" ninst
        |1 →5 "|  Nombre de mots                   :          |" nword |1 →5
        "|  Nombre de champs continuation :          |" ncont |1 →5 "|"
        →53 "|"))
    (IF ??
       (TABSTAT "Occurrences statiques des instructions" ~linstr '!inst
        ninst 0 [" Nombre d'instructions          : " ninst]))
    (IF ??
       (TABSTAT "Occurrences statiques des formats" ~lformat '!format
        ninst 0
        [" Nombre d'instructions         : "
         ninst
         " Nombre de mots                : "
         nword]))
    (IF ??
       (TABSTAT "Occurrences statiques des continuations" ~lcont '!contin
        ncont ninst
        [" Nombre de champs continuation : "
         ncont
         " Nombre d'instructions         : "
         ninst]))
    (IF ??
       (PROGN
          (SETQ
             x
              (APPLY 'PLUS
                (MAPCT ~loper (LAMBDA (x) (NUMBP (GET x '!operand))))))
          (TABSTAT "Occurrences statiques des opérandes" ~loper '!operand
           x ninst
           [" Nombre d'opérandes visités     : "
            x
            " Nombre d'instructions          : "
            ninst])))
    (IF ??
       (TABSTAT "Occurrences statiques des SUBR" ~ltypsubr '!typsubr
        nfunct 0 [" Nombre de fonctions   SUBR    : " nfunct]))
    (IFN ? (LESCAPE) (TERPRI 2))
    (SETQ nfunct 0)
    (SETQ
       l (SORTL lfunct)
       lt
       '((1 . 0SUBR) (2 . 1SUBR) (3 . 2SUBR) (4 . 3SUBR) (5 . NSUBR)
         (6 . FSUBR)))
    (WHILE l
       (PRINT (INCR nfunct) →5 (CAR l) →17
```

```
        (CASSQ (GET (NEXTL l) '!ftyp) lt) →30 (INCR nfunct) →35 (CAR l)
        →47 (CASSQ (GET (NEXTL l) '!ftyp) lt))))

; Epilogue standard ;
; ————————————————— ;
;;
; remet la table de lecture précédente ;

(READ.TABLE READ.TABLE)

(DE HOT.START ()
   (STATUS 2 34)
   (STATUS 1 1)
   (STATUS 2 27)
   "# Simulateur de la machine VCMC2 chargé.")

(HOT.START)
```

CROSS REFERENCE

Signification des codes associés aux numéros des lignes :

- \# définition de fonction de type DE DF DM DMC ou ENTRY
- & définition de fonction de type ESCAPE, ESCLOOP
- : définition d'étiquette dans PROG, DO, LAP ...
- , variable argument d'une fonction
- = variable affectée par SETQ ou SETQQ
- ' nom apparaissant dans une S-expression quotée
- \- code instruction assembleur

→	DMC	62
→	DMO	60
→	DE	58
ANACODE	DE	865
CODE	DF	53
HOT.START	DE	1058
JUMP	DE	426
MACHERR	DE	409
MOVE	DE	421
POINT	DE	335
PRININST	DE	114
PRINTL	DE	99
PRSTACK	DE	435
SORTL	DE	75
TABSTAT	DE	815
VALUESOURCE	DM	331
VCMC2	DE	204
XCT	DE	461
|	DMC	70
|	DMO	68
|	DE	66

1	→	58#	58'	60#	62#	62'		
2	→10	794						
3	→13	853						
4	→14	117						
5	→15	403						
6	→17	1047						
7	→22	853						
8	→30	1048						
9	→32	853						
10	→35	1048						
11	→37	854						
12	→42	854						
13	→47	854	1049					
14	→5	117	1001	1003	1004	1004	1005	1047
15	→51	827	827	832	833	854	855	
16	→53	1006						
17	→55	827						
18	→6	413	415					
19	→8	827	853					
20	!contin	251'	300'	772'	885'	994'	1018'	
21	!cval	583'	590'					
22	!format	250'	292'	504'	884'	1000'	1011'	

23	!ftyp	517'	605'	608'	929'	1048'	1049'					
24	!fval	512'	599'	602'	924'							
25	!inst	249'	289'	491'	883'	972'	1008'					
26	!label	272'	273'	317'	319'	320'						
27	!operand	252'	306'	344'	347'	352'	373'	374'	886'	979'	981'	985'
		989'	1029'	1030'								
28	!plist	593'	596'									
29	!point	390'	400'									
30	!ptyp	611'	614'	930'								
31	!typsubr	887'	931'	1037'								
32	#nbcons	225,	243=	283	548	550=	619	623				
33	#ncont	222,	300	302	772							
34	#nincr	221,	240=	466	547	550=						
35	#ninst	220,	239=	280	282	290	290	293	295	300	304	307
		311	320	324	465	467	471					
36	#noper	223,	241=	307	309	339						
37	#nword	224,	242=	297	501=	501						
38	#stackl	226,	244=	284	359	360=	549	550=	780	780=		
39	&	446'										
40	&INTERN	723&	726									
41	&STOP	258&	763									
42	/(	106'	449'									
43	/)	112'	452'	454'								
44	*	846	847	851	852							
45	**	802'	806'									
46	*VAL*	388'										
47	*eos*	237'	366'									
48	,	118'	120'									
49	-	282										
50	/...	105'	111'									
51	/.../)	454'										
52	//	282	846	847	851	852						
53	0SUBR	168'	515'	927'	1044'							
54	1MOT	144'	503'	999'								
55	1SUBR	168'	515'	927'	1044'							
56	2MOTS	144'	503'	999'								
57	2SUBR	168'	515'	927'	1044'							
58	3MOTS	144'	503'	999'								
59	3SUBR	168'	515'	928'	1044'							
60	4MOTS	144'	503'	999'								
61	:	116'	448'									
62	<	106'										
63	=	403'	794'	802'	806'							
64	>	112'	847	852								
65	?	867,	1039									
66	??	868,	883	884	885	886	887	931	965	1002	1007	1010
		1017	1024	1036								
67	?contrace	187'	187=	472	521=	523=	525=	527=	529	532=	541=	787
68	?instat	207,	249	250	251	252	288	291	298	305	339	344
		347	352	371	426,	491	492	771				
69	?instrace	191'	191=	260	521=	523=	525=	527=	529	531=	542=	
70	?labstat	208,	270	315								
71	?pointrace	183'	183=	403	529	534=	539=					
72	?prinstack	195'	195=	521=	523=	525=	527=	529	533=	540=	798	
73	?stat	206,	278	463	545							
74	?stepper	199'	199=	261	525=	527=	529	535=	538=			
75	@	372'	378'	982'	986'							
76	A1	140'	148'	211,	233=	326	343'	472	980'			
77	A2	140'	148'	212,	234=	343'	472	980'				
78	A3	140'	148'	213,	235=	343'	472	980'				
79	A4	140'	148'	214,	236=	343'	472	980'				
80	ADD	161'	641									
81	ADD1	107	108	109	110	273						

82	ANACODE	865#										
83	AND	111	358	377	467	498	771	773	780	787	948	952
		982	986									
84	APPEND	470										
85	APPLY	722	1028									
86	ASCII	727	738									
87	ATOM	102	116	353	369	443	476	694	898			
88	BLOCK	518	932'	969'								
89	BOUNDP	128	132	136	183	187	191	195	199			
90	C	749	750									
91	CAADR	385										
92	CAAR	725	901	906	908	910	912	917	932	948	952	958
93	CADAR	725	914	919	921	922	923	925	930			
94	CADDAR	926	931									
95	CADDDR	930										
96	CADDR	235	938									
97	CADR	234	333	376	382	383	384	453	575	586	727	776
		781	935									
98	CALL	152'	417'	500'	777	995'						
99	CAR	93	93	159'	233	260	266	270	272	273	333'	366
		372	376	377	378	388	390	394	395	396	400	403
		442	448	450	500	555	555	557	558	568	573	590
		596	602	608	614	633	635	737	772	774	791	791
		792	792	795	898	902	907	909	911	913	930	935
		938	944	956	961	982	986	994	995	1047	1048	
100	CASCII	263	471'	718	750							
101	CASSOC	503	998									
102	CASSQ	514	926	1048	1049							
103	CDADR	387										
104	CDAR	750	907	909	911	948	952	958				
105	CDDAR	896	914	923								
106	CDDR	454										
107	CDR	89	95	97	159'	377	377	389	451	456	556	55
		570	720	747	751	939						
108	CNTH	567	572	681	684							
109	CODE	53#										
110	COND	342	375	684	736	905	916	978				
111	CONS	160'	618									
112	CVAL	159'	580									
113	DATA	376'	385'	518	568'	573'	725'	938'	969'			
114	DE	58	66	75	99	114	204	335	409	421	426	43
		461	815	865	1058							
115	DF	53										
116	DIFFER	284	646	705	710							
117	DISPLAY	469										
118	DISPT	162'	682									
119	DIV	161'	651									
120	DM	331										
121	DMC	62	70									
122	DMO	60	68									
123	ENTRY	511	917'	969'								
124	EPROGN	389										
125	EQ	162'	263	264	351	366	372	376	377	378	388	69
		697	791	906	907'	908	910	912	917	932	986	
126	EQN	85										
127	EQUAL	792										
128	ESCAPE	258	723									
129	EVAL	265										
130	EVALA1	317'										
131	EXPLODE	471	746									
132	FATOM	162'	692									
133	FIX	847	852									
134	FLIST	162'	694									

135 FLOAT	282	836	836	846	851						
136 FNIL	908'										
137 FNUMB	162'	696									
138 FSUBR	168'	516'	928'	1045'							
139 FTYP	160'	603									
140 FVAL	160'	597									
141 GE	104	162'	361	447	702	702					
142 GENSYM	722'										
143 GET	272	317	319	390	593	599	605	611	840	1029	1048
	1049										
144 GETL	583										
145 GT	162'	359	701	701	780	902					
146 HOT.START	1058#	1064									
147 IF	78	85	93	102	104	107	108	109	110	116	249
	250	251	252	260	264	266	270	273	278	288	291
	298	305	315	319	339	344	347	352	353	354	361
	366	371	372	385	403	414	429	438	440	443	445
	447	451	454	463	467	472	476	491	492	495	496
	497	568	573	585	587	725	742	746	771	773	787
	828	841	844	847	849	852	883	884	885	886	887
	898	899	919	931	936	948	952	956	958	969	991
	995	1002	1007	1010	1017	1024	1036				
148 IFN	120	261	348	397	487	545	798	965	1039		
149 IN	163'	717									
150 INCR	339	465	466	495	496	500	619	623	772	853	918
	970	971	983	984	987	988	993	996	996	1047	1048
151 INDEX	159'	218,	562	563	564=	567	572	802'	803		
152 INTERN	163'	719									
153 IRCAMP	467										
154 JUMP	152'	426#	433'	500'	681	684	776	776	781	782	
155 JUMPX	161'	679									
156 L	75,	78	81	81	249	249	250	250	251	251	252
	252	409,	413	883	883	884	884	885	885	886	886
	887	887									
157 L1	82	84=	85	87	88=	88	89	90	91=	91	92
	93	94	94=	94	95	95=	95				
158 L2	75,	91=	93	94	94=	96					
159 LAMBDA	249	250	251	252	319	587'	749	789	839	883	884
	885	886	887	892	897	976	1029				
160 LCOD	205,	238									
161 LE	162'	700	700								
162 LENGTH	81	357	447	779	824	824	902				
163 LESCAPE	85	116	438	441	476	841	898	903	965	969	1039
164 LET	82	255	442	724	837	935	944				
165 LEZP	936										
166 LIST	120'										
167 LISTP	259	266	429	495	496	499	585	693	956	982	986
168 LITATOM	487	685	691	692	901						
169 LL	75,	91=	94	95=	96						
170 LOGAND	89	161'	665	666							
171 LOGOR	161'	662	663								
172 LOGSHIFT	86										
173 LOGXOR	161'	668	669								
174 LT	162'	699	699								
175 LZP	467										
176 MACHERR	350	353	362	367	369	386	398	402	409#	433	476
	487	519	569	574	767	783					
177 MAKLIST	737	746									
178 MAP	896										
179 MAPC	249	250	251	252	748	788	838	883	884	885	886
	887	891	975								
180 MAPCAR	471										
181 MAPCT	319	1029									

182 MEMQ	343	382	383	384	394	395	396	500	587	787	919
	923	948	952	958	969	980	995				
183 MERROR	417'										
184 MOVE	159'	421#	554	554	911'	914'					
185 MOVEX	159'	565									
186 MUL	161'	648									
187 N	75,	84=	85	86	89	94=	94	94=			
188 NCONC	934										
189 NCONS	948'	952'	958'								
190 NEQ	162'	385	568	573	698	698	725	909'	982		
191 NEWL	727	921									
192 NEXTL	107	108	109	110	117	117	119	268	271	274	368
	482	483	484	727	797	832	832	946	950	954	1048
	1049										
193 NOP	152'	161'	417'	675	775						
194 NOT	787	901									
195 NSUBR	168'	516'	928'	1044'							
196 NTH	88										
197 NULL	78	270	270	346	429	438	451	454	692	696	712
	979										
198 NUMBP	686	695	696	738	834	835	840	1029			
199 OBLIST	319										
200 OR	128	132	136	183	187	191	195	199	270	381	393
	429	512	725	790	834	835	840	844	849	900	
201 OUT	163'	733									
202 OUTBUF	758										
203 OUTCH	752										
204 PC	215,	238=	259	260	266	268	270	271	272	273	274
	275	431=	778								
205 PLEN	163'	740									
206 PLENGTH	742										
207 PLIST	159'	591									
208 PLUS	89	501	567	572	642	681	824	1028'			
209 PNAM	163'	743									
210 POINT	333'	335#	341=	350'	353'	362'	367'	369'	386'	398'	402'
	403'	403	404	424	555	556	557	558	563	575	584
	590	593	596	599	602	605	608	611	614	621	625
	629	633	637	643	647	650	654	658	664	667	670
	706	711	716	718	721	742					
211 POP	84	84	538	539	540	541	542	757	910'		
212 PRATOM	752										
213 PRIN1	103	105	106	111	112	116	117	119	120	413	444
	446	448	449	452	454	454	731	756	794	802	806
214 PRINC	116	118	120	735							
215 PRINI	163'	731									
216 PRININST	114#	260									
217 PRINT	265	275	280	281	283	284	403	412	414	415	547
	548	549	824	832	833	853	855	889	903	1001	1003
	1047										
218 PRINTL	99#	107	108	109	110	795	803	807			
219 PRINTLEVEL	809										
220 PROG1	368										
221 PROGN	49	263	279	316	356	464	493	772	793	829	992
	996	1025									
222 PRSTACK	163'	435#	762	762							
223 PRSTAT	544										
224 PTYP	160'	609									
225 PUSH	81	87	89	529	753	912'					
226 PUT	249	250	251	252	273	400	512	513	590	596	602
	608	614	883	884	885	886	887	922	925	930	
227 PUTINCR	344	347	352	373	374	491	502	772	931	972	979
	981	985	989	994	997						
228 Q	75,	86=	87	88	89						

229 QUO	653										
230 QUOTE	377										
231 R1	256	421,	423=	423	424	426,	428=	428	429	429	431
	433	483=	495	512	512	513	554	555=	555	555	556=
	556	556	557=	557	557	558=	558	558	563	564	566=
	566	570	572=	572	573	575	582=	582	583	587	587
	589=	589	590	592=	592	593	595=	595	596	598=	598
	599	601=	601	602	604=	604	605	607=	607	608	610=
	610	611	613=	613	614	620=	620	621	624=	624	625
	629	631	633	636	637	642=	642	643	645=	645	646=
	646	647	649=	649	650	652=	652	653=	653	654	656=
	656	657=	657	658	663=	663	664	666=	666	667	669=
	669	670	680=	680	681	683=	683	687	691	692	693
	694	695	696	697	698	699	700	701	702	704=	704
	705=	705	706	707	709=	709	710=	710	711	712	716
	718	720=	720	725	725	727	731	734=	734	737	737
	738	738	739	741=	741	742	742	742	745=	745	746=
	746	746	746	748	756	762					
232 R2	256	421,	424	484=	496	514	554	555	556	557	558
	567=	567	568	570	575	584	590	593	596	599	602
	605	608	611	614	620	621	624	625	642	643	646
	647	649	650	653	654	657	658	663	664	666	667
	669	670	681	683=	683	685	686	697	698	699	700
	701	702	705	706	710	711	721	742	747=	747	750
	751=	751									
233 READ	62	70	163'	265	716	716					
234 READ.STD	49										
235 READ.TABLE	49=	49	1056	1056							
236 READCH	718										
237 REM	161'	467	655	657							
238 RESL	75,	80=	91	97							
239 RETURN	152'	782	995'								
240 REVERSE	726										
241 REVIVE	536										
242 RPLACA	557	750									
243 RPLACB	333	907	909	911	913	933	961				
244 RPLACD	90	91	94	96	558						
245 RUNTIME	245	282									
246 SCAR	159'	557									
247 SCDR	159'	558									
248 SCVAL	159'	588									
249 SELECTQ	507	774									
250 SELF	91	91	376	389	450	453	456	727	939		
251 SET	424	555	556	563	570	575	584	593	599	605	611
	621	625	629	633	636	637	643	647	650	654	658
	664	667	670	706	711	716	718	721	742		
252 SETQ	49	80	84	86	88	91	94	95	128	132	136
	140	144	148	152	156	168	172	183	187	191	195
	199	232	272	317	340	357	360	363	368	379	391
	399	423	428	431	472	480	494	501	521	523	525
	527	530	537	543	550	555	556	557	558	564	566
	567	572	582	583	589	592	595	598	601	604	607
	610	613	620	624	631	635	642	645	646	649	652
	653	656	657	663	666	669	680	683	691	692	693
	694	695	696	697	698	699	700	701	702	704	705
	707	709	710	712	720	734	741	744	751	778	779
	780	834	835	836	840	842	882	945	974	1026	1040
	1041										
253 SFTYP	160'	606									
254 SFVAL	160'	600									
255 SILENCE	528										
256 SINDEX	159'	564									
257 SORT	93										

258 SORTL	75#	158	319	1042							
259 SPLIST	160'	594									
260 SPTYP	160'	612									
261 SSTACK	160'	630	787'								
262 ST	217,	237=	353	357	363=	363	364	366	368	368	629
	631=	633	635	636	762	778=	778	779	806'	807	
263 STACK	160'	629	787'								
264 STATUS	43	753	755	757	800	808	823	858	1059	1060	1061
265 STEP	524										
266 STOP	164'	237'	763								
267 STRINGL	742										
268 STRINGP	737	742	746								
269 SUB	161'	644									
270 SUB1	450	453	456	939							
271 SUBFZ	163'	708									
272 SUBTZ	163'	703									
273 T	370	402	424	486	521	521	521	525	525	525	525
	555	556	557	558	563	575	584	587'	590	593	596
	599	602	605	608	611	614	621	625	629	633	637
	643	647	650	654	658	664	667	670	687	706	711
	716	718	721	739	742	940	990				
274 TABSTAT	289	292	299	306	318	815#	1008	1011	1018	1030	1037
275 TATOM	162'	691									
276 TERPRI	68	116	121	163'	414	455	732	732	758	796	804
	810	1039									
277 TIMES	649										
278 TLIST	162'	693									
279 TNIL	906'										
280 TNUMB	162'	695									
281 TOPST	161'	632									
282 TRACE	520										
283 TRMOP	467										
284 TST	148'	351'	363=	363	368=	782'	911'	914'	980'		
285 TTAB	60	439	801	805							
286 TYI	263	264									
287 TYO	263										
288 TYS	264										
289 UNSTEP	526										
290 UNTIL	263										
291 UNTRACE	522										
292 V	409,	412									
293 VALUESOURCE	331#	423	428	555	556	557	558	564	566	567	572
	582	589	592	595	598	601	604	607	610	613	620
	620	624	624	631	636	642	642	645	646	649	649
	652	653	656	657	663	663	666	666	669	669	680
	681	683	683	691	692	693	694	695	696	697	697
	698	698	699	699	700	700	701	701	702	702	704
	705	709	710	720	731	734	741	745	747	756	762
294 VCMC2	204#										
295 WHILE	92	259	831	1046							
296 XCONS	160'	622	624								
297 XCT	268	417	461#	476'	487'	519'	569'	574'	767'	783'	
298 XMOVE	159'	571									
299 XTOPST	161'	634									
300 ZEROP	440	445	467	707	712	725	841	844	844	849	849
301 ↑	263'										
302 adef	335,	343	344	345	346	351	352	369	369	372	376
	376	377	377	377	378	382	383	384	388	389	390
	394	395	396	400	402	403					
303 clair	817,	824	838								
304 codecondition	256	486=	691=	692=	693=	694=	695=	696=	697=	698=	699=
	700=	701=	702=	707=	712=	773					
305 codop	946=	961	969	972							

306	cont	878,	955=	961	991	994	995					
307	continuation	256	485=	499	500	512	543=	771	772	773	774	776
		781										
308	d	442	445	450	453							
309	direct	148'	373'	989'								
310	gtot	820,	835	835=	836=	836	849	851				
311	i	114,	116	116	117	117	119	120	120	875,		
312	idest	335,	348	354	389							
313	immediat	148'	374'	985'								
314	indic	818,	840									
315	inst	256	482=	487	491	507	787	897	898	901	902	903
		906	907	907	908	909	909	910	911	911	912	913
		914	914	917	919	921	922	923	925	926	930	930
		931	932	933	935	938	939	944	961			
316	l	331,	333	333	481=	482	483	484	485	724	726	727
		944	946	948	948	950	952	952	954	956	958	958
		960	1042=	1046	1047	1048	1048	1049				
317	larg	209,	233	234	235	236						
318	laux	53,	53	839	840	853						
319	lcomm	821,	824	828	831	832	832					
320	lfunct	870,	919	921	1042							
321	linst	415	461,	476	481	892	896	923				
322	llinst	866,	889	891								
323	lt	1043=	1048	1049								
324	n	99,	104	106	107	108	109	110	112	435,	440	456
		935	936	939								
325	nbaux	58,	58	60,	60	66,	66	68,	68	837	853	
326	ncont	873,	882=	993	1005	1019	1021					
327	nfunct	871,	882=	918	1001	1038	1038	1040=	1047	1048		
328	ninst	872,	882=	970	1003	1009	1009	1012	1014	1019	1023	1031
		1035										
329	null	148'	347'	979'								
330	nword	882=	971	984	988	996	1004	1016				
331	op	976	979	980	981	982	982	986	986			
332	op1	876,	947=	961	975							
333	op2	877,	951=	961	975							
334	perc1	839	843=	847=	847	847	854					
335	perc2	839	848=	852=	852	852	854					
336	runtime	227,	245=	282								
337	s	99,	102	103	107	107	108	108	109	109	110	110
		111	442	443	444	447	448	450	451	453	454	
338	stack	435,	438	442	456							
339	titre	816,	827									
340	total	819,	834	834=	836=	836	844	846				
341	valregs	256	472=	791	792	797						
342	x	228,	272=	273	273	317=	319	319	319	320	322	494=
		495	496	500	501	503	974=	983	987	996	998	1027=
		1029	1029	1031	1033							
343	xaux	256	335,	357=	359	360	361	380=	385	387	392=	397
		399=	399	400	401	409,	414	414	583=	585	586	635=
		637	779=	780	780	789	791	792	794	795	839	840=
		841	844	846	849	851	853					
344	{	106'										
345	|	66#	66'	68#	70#	70'						
346	|1	824	824	824	825	826	827	827	855	856	857	1003
		1004	1004	1005								
347	}	112'										
348	~UNDEF	587'										
349	~interpreter	128'	128=	384	396							
350	~lcont	152=	251	299	885	1018						
351	~lformat	144=	250	292	884	1011						
352	~linstr	157=	249	289	883	1008						
353	~loper	148=	252	306	886	1029	1030					

354	~lregn	140=	788		
355	~ltypsubr	168=	887	1037	
356	~pprinter	132'	132=	383	395
357	~reader	136'	136=	382	394
358	~stacklength	172=	361		

APPENDICE C

```
;                    V C M C 2 R   .   V L I                        ;
;                                                                   ;
;               LECTEUR  VLISP  en machine VCMC2                    ;
;-------------------------------------------------------------------;
;          Ce fichier doit être édité au moyen de :                 ;
;              (CROSSF V2R T T T {NIL/T} T T)                       ;
;                {NIL/T} si VERSATEC                                ;
;-------------------------------------------------------------------;
;                                                                   ;
;                     Jérôme CHAILLOUX                              ;
;                                                                   ;
;          Département d'Informatique                               ;
;          Université de Paris 8 - Vincennes                        ;
;          Route de la Tourelle 75571 Paris Cédex 12                ;
;          Tél : 374 12 50 poste 299                                ;
;                                                                   ;
;          L.I.T.P.   (CNRS LA 248)                                 ;
;          2 Place Jussieu 75221 Paris Cédex 05                     ;
;          Tél : 336 25 25 poste 53-70                              ;
;                                                                   ;
;          I.R.C.A.M.                                               ;
;          31 Rue St Merri 75004 Paris                              ;
;          Tél : 277 12 33 poste 48-48                              ;
;-------------------------------------------------------------------;

(STATUS 2 1 2)

; Pour prettyprinter correctement le code ;

(DF CODE (L) L)


; Pour contrôler la lecture ;

(DMC "S" () (OUTSTR (STRING (READ))) '(NOP))
```

```
; Imprimeur VLISP ;
;------------------;

(RPLACA '~reader
 (CODE NIL

         ;┌──────────────────────────────────────────┐;
         ;│                                          │;
         ;│          Lecteur VLISP - VCMC2           │;
         ;│       (à utiliser avec  V2I.VLI)         │;
         ;│                                          │;
         ;└──────────────────────────────────────────┘;

         ; Initialise le lecteur ;

READINI:MOVE    '0,(@ RINGUR)        ; rien à relire ;
        MOVE    NIL,(@ IMPLI),[RETURN]; dans READ ;


         ; Accès au caractère physique ;
         ; doit mettre dans A4 le caractère suivant ;
         ;--------------------------------;


         ; Le plus simple ;

INCHB:  IN      A4,,[RETURN]
```

```
                ; Accès au caractère logique ;
                ;------------------------------;


                ; GETCH : lit le caractère suivant ou celui à reingurgiter ;
                ;         dans le flux d'entrée courant ;
                ;     retourne A4 ← le caractère ;
                ;              A3 ← le type du caractère (type-ch) ;
                ;            type-ch = 0 :  caractères null (à ignorer) ;
                ;            type-ch = 1 :  début de commentaires  ;
                ;            type-ch = 2 :  fin de commentaires ;
                ;            type-ch = 3 :  quote caractère ;
                ;            type-ch = 4 :  début de liste ;
                ;            type-ch = 5 :  fin de liste ;
                ;            type-ch = 6 :  début de liste crochée ;
                ;            type-ch = 7 :  fin de liste crochée ;
                ;            type-ch = 8 :  point ;
                ;            type-ch = 9 :  séparateur nul ;
                ;            type-ch = 10 : macro-caractère ;
                ;            type-ch = 11 : délimiteur de chaîne de caractères ;
                ;            type-ch = 12 : caractère normal ;

        GETCH:  EQ      (@ RINGUR),'0,[JUMP (GETCH1)] ; vide ;
                MOVE    (@ RINGUR),A4          ; recupère le caractère ;
                MOVE    '0,(@ RINGUR),[JUMP (GETCH2)] ; et l'efface ;

        GETCH1: FNIL    (@ IMPLI),,[JUMP (GETCH3)] ; on se trouve dans IMPLODE ;
                NOP     ,,[CALL (INCHB)]       ; lecture physique ;
        GETCH2:                                ;** recherche du type ;
                SINDEX  (TABCH)                ; charge le registre d'index ;
                XMOVE   A4,A3,[RETURN]         ; A3 ← le type ;

        GETCH3: MOVE    (@ IMPLD),A4           ; récupère la liste IMPLODE ;
                TNIL    A4,,[JUMP (GETCH6)]    ; elle est vide ;
                FLIST   A4,,[JUMP (ERLEC1)]    ; c'est une erreur ;
                CDR     A4,(@ IMPLD)           ; sauve le reste ;
                CAR     A4,A4                  ; A4 ← le caractère suivant ;
                PNAM    A4,(BUFCH)             ; BUFCH ← PNAME du caractère ;
                MOVE    (@ BUFCH),A4,[JUMP (GETCH2)]
                                               ; A4 ← le 1er caractère du Pname ;
        GETCH6: MOVE    'T,(@ IMPLD)           ; marque liste vide ;
                MOVE    '" ",A4,[JUMP (GETCH2)] ; séparateur normal ;

```

```
                ; Accès au caractère logique valide ;
                ;-------------------------------------;

                ; GETCV : retourne dans A4 le caractère LISP valide suivant ;
                ;         traite les commentaires et les caractères quotés ;
                ;         retourne dans A3 le type du car. valide (type-cv) ;
                ;                 type-cv = 0 : ( ;
                ;                 type-cv = 1 : ) ;
                ;                 type-cv = 2 : [ ;
                ;                 type-cv = 3 : ] ;
                ;                 type-cv = 4 : . ;
                ;                 type-cv = 5 : séparateur ;
                ;                 type-cv = 6 : macro-caractère ;
                ;                 type-cv = 7 : délimiteur de chaîne ;
                ;                 type-cv = 8 : normal P-name ;

GETCV1: NOP     ,,[CALL (GETCH)]            ; nouveau caractère ;
        EQ      A3,'1,[JUMP (GETCV)]        ; si nouveau début commentaires ;
        NEQ     A3,'2,[JUMP (GETCV1)]       ; si non-fin commentaires ;
GETCV:                                      ;** point d'entrée ;
        NOP     ,,[CALL (GETCH)]            ; lit un caractère ;
        EQ      A3,'0,[JUMP (GETCV)]        ; saute tous les nulls ;
        NEQ     A3,'3,[JUMP (GETCV2)]       ; test quote caractère ;
        NOP     ,,[CALL (GETCH)]            ; relit un nouveau caractère ;
        MOVE    '8,A3,[RETURN]              ; force le type normal - 4 ;

GETCV2: EQ      A3,'1,[JUMP (GETCV1)]       ; si début commentaires ;
        EQ      A3,'2,[JUMP (GETCV)]        ; ignore les fin de commentaires ;
        SUB     '4,A3,[RETURN]              ; retourne type - 4 ;
```

```
          ; Accès à l'unité syntaxique suivante ;
          ;-------------------------------------;


          ; RD1 : lit l'unité syntaxique suivante ;
          ;       retourne dans A3 son type : ;
          ;          type-us   = 0 :       (        ;
          ;          type-us   = 1 :       )        ;
          ;          type-us   = 2 :       [        ;
          ;          type-us   = 3 :       ]        ;
          ;          type-us   = 4 :       .        ;
          ;          type-us   = 5 :    objet VLISP ;
          ;              (et A1 contient l'objet) ;

RD1:      NOP      ,,[CALL (GETCV)]        ; caractère suivant ;
          JUMPX    (RDTB1),A3              ; aiguillage sur le type du caractère ;

RDTB1:    DATA     (RDPARO)                ; type-cv = 0 : ( ;
          DATA     (RDPARF)                ; type-cv = 1 : ) ;
          DATA     (POPJ)                  ; type-cv = 2 : [ ;
          DATA     (POPJ)                  ; type-cv = 3 : ] ;
          DATA     (POPJ)                  ; type-cv = 4 : . ;
          DATA     (RD1)                   ; type-cv = 5 : séparateur ;
          DATA     (RDMAC)                 ; type-cv = 6 : macrocaractère ;
          DATA     (RDSTR)                 ; type-cv = 7 : délimiteur de chaînes ;
          DATA     (RD2)                   ; type-cv = 8 : normal P-name ;

RDPARO:                                    ; traitement ( ;
          ADD      '1,(@ RDPRD),[RETURN]   ; comptage du pretty-read ;

RDPARF:                                    ; traitement ) ;
          SUB      '1,(@ RDPRD)
          GE       (@ RDPRD),'0,[RETURN]   ; correct ;
          MOVE     '0,(@ RDPRD),[RETURN]   ; force un 0 ;

RDMAC:                                     ; traitement des macro-caractères ;
          MOVE     A4,A1,[CALL (ASCII)]    ; convertit le caractère en atome ;
          MOVE     NIL,A2,[CALL (APPLY)]   ; et appelle la fonction ;
          MOVE     '5,A3,[RETURN]          ; type-us = objet VLISP ;

RDSTR:                                     ; traitement des chaînes de caractères ;
          MOVE     '0,A2                   ; init index sur BUFAT ;
RDSTR1:   NOP      ,,[CALL (GETCH)]        ; caractère simple suivant ;
          EQ       A3,'11,[JUMP (RDSTR2)]  ; c'est la fin ;
          SINDEX   (BUFAT)                 ; init INDEX ;
          MOVEX    A4,A2                   ; charge le nouveau caractère ;
          ADD      '1,A2                   ; avance l'indice ;
          LT       A2,'30,[JUMP (RDSTR1)]  ; il est correct ;
          MOVE     '2,A1,[JUMP (ERLEC)]    ; sinon c'est une erreur ;
RDSTR2:   SINDEX   (BUFAT)                 ; repositionne l'index ;
          MOVEX    '0,A2                   ; force le dernier 0 ;
          INTERN   (BUFAT),A1              ; interne l'atome ;
          SCVAL    A1,A1                   ; qui devient une constante ;
          MOVE     '5,A3,[RETURN]          ; type-us = objet VLISP ;

RD2:                                       ; traitement atome normal ;
          MOVE     '0,A2                   ; init indice sur BUFAT ;
RD21:     SINDEX   (BUFAT)                 ; init INDEX ;
          MOVEX    A4,A2                   ; charge le caractère dans BUFAT ;
          ADD      '1,A2                   ; avance l'indice ;
          GE       A2,'30,[JUMP (ERLEC3)]  ; il n'est plus correct ;
          NOP      ,,[CALL (GETCV)]        ; caractère valide suivant ;
```

```
        EQ      A3,'8,[JUMP (RD21)]         ; si type-cv = normal ;
        MOVE    A4,(@ RINGUR)               ; pour le relire ;
        SINDEX  (BUFAT)                     ; repositionne l'INDEX ;
        MOVEX   '0,A2                       ; force le dernier 0 ;
        INTERN  (BUFAT),A1                  ; interne l'atome ;
        MOVE    '5,A3,[RETURN]              ; type-us = objet VLISP ;

        ; Macro-caractère standard ;

        ENTRY   /',0SUBR

        ;         '<s> = (QUOTE <s>) ;

/':     NOP     ,,[CALL (READI)]            ; lit l'exprssion <s> ;
        XCONS   NIL,A1                      ; forme (<s>) ;
        CONS    'QUOTE,A1,[RETURN]          ; puis (QUOTE <s>) ;

        ; Définition d'un macro-caractère ;

        ENTRY   DMC,FSUBR,1

DMC:    NOP     ,,[CALL (DE)]               ; défini la fonction ;
        PUSH    A1                          ; sauve le caractère ;
        MOVE    '10,A2,[CALL (TYPECH)]; type-ch = macro caractère ;
        POP     A1,,[RETURN]                ; et c'est tout ;
```

```
            ; Accès à l'expression ;
            ;-------------------------;


            ; READI : lecture interne ;
            ; retourne dans A1 l'expression lue suivante ;

READI:   NOP     ,,[CALL (RD1)]          ; us suivante ;
READ0:   JUMPX   (READT1),A3             ; aiguillage sur le 1er type-us ;

READT1:  DATA    (READ2)                 ; type-us = 0 : ( ;
         DATA    (ERLEC4)                ; type-us = 1 : ) ;
         DATA    (READ1)                 ; type-us = 2 : [ ;
         DATA    (ERLEC4)                ; type-us = 3 : ] ;
         DATA    (ERLEC4)                ; type-us = 4 : . ;
         DATA    (POPJ)                  ; type-us = 5 : objet ;

            ; un [ a été lu ;

READ1:   NOP     ,,[CALL (RD1)]          ; us suivante ;
         EQ      A3,'3,[JUMP (FALSE)]    ; c'est donc []  => NIL ;
         MOVE    NIL,A2                  ; fabrique une tête de liste ;
         CONS    '*MARK*,A2              ; (*MARK*) ;
         PUSH    A2                      ; sauve la tête de liste ;
         PUSH    A2,,[JUMP (READ31)]     ; traite l'us ;

            ; une ( a été lue ;

READ2:   NOP     ,,[CALL (RD1)]          ; us suivante ;
         EQ      A3,'1,[JUMP (FALSE)]    ; ()  => NIL ;
         NOP     ,,[CALL (READ0)]        ; 1er élément ;
         XCONS   NIL,A1                  ; fabrique le 1er doublet ;
         PUSH    A1                      ; sauve le 1er doublet ;
READ3:                                   ; traite l'us suivante ;
         PUSH    A1,,[CALL (RD1)]        ; us suivante ;
READ31:                                  ; elle est prête ;
         JUMPX   (READT2),A3             ; aiguillage sur le type-us ;

READT2:  DATA    (READ5)                 ; type-us = 0 :  (  ;
         DATA    (READ7)                 ; type-us = 1 :  )  ;
         DATA    (READ6)                 ; type-us = 2 :  [  ;
         DATA    (READ8)                 ; type-us = 3 :  ]  ;
         DATA    (READ9)                 ; type-us = 4 :  .  ;
         DATA    (READ4)                 ; type-us = 5 : objet ;

            ; rajoute l'objet VLISP contenu dans A1 ;

READ4:   XCONS   NIL,A1                  ; nouvel élément ;
         POP     A2                      ; récupère LAST ;
         SCDR    A1,A2,[JUMP (READ3)]    ; et c'est tout ;

            ; traite une ( ;

READ5:   PUSH    (READ4),,[JUMP (READ2)] ; JRST HACK ;

            ; traite un [ ;

READ6:   PUSH    (READ4),,[JUMP (READ1)] ; JRST HACK ;

            ; traite une ) ;

READ7:   POP     A1                      ; dernier doublet ;
```

```
READ71: POP     A1                          ; 1er doublet ;
        CAR     A1,A2                       ; A2 pour tester la marque ;
        NEQ     A2,'*MARK*,[RETURN]         ; s'il n'y a pas de marque ;
        MOVE    '5,A1,[JUMP (ERLEC)]        ; probablement [ ... ) ;

        ; traite un ] ;

READ8:  POP     A1                          ; dernier doublet ;
        POP     A1                          ; 1er doublet ;
        CAR     A1,A2                       ; A2 pour tester la marque ;
        NEQ     A2,'*MARK*,[JUMP (ERLEC6)]  ; il manque la marque ;
        SCAR    'LIST,A1,[RETURN]           ; fabrique  (LIST ...) ;

        ; traite un . ;

READ9:  NOP     ,,[CALL (READI)]            ; lecture de l'expression ;
        PUSH    A1,,[CALL (RD1)]        ; sauve sa val et lit le séparateur ;
        POP     A1                          ; dernière expression ;
        POP     A2                          ; dernier doublet ;
        JUMPX   (READT3),A3           ; aiguillage sur le type dernière us ;

READT3: DATA    (ERLEC7)                    ; type-us = 0 : ( ;
        DATA    (READ91)                    ; type-us = 1 : ) ;
        DATA    (ERLEC7)                    ; type-us = 2 : [ ;
        DATA    (READ92)                    ; type-us = 3 : ] ;
        DATA    (ERLEC7)                    ; type-us = 4 : . ;
        DATA    (ERLEC7)                    ; type-us = 5 : objet ;

READ91:                                     ; cas :  . <s> ) ;
        SCDR    A1,A2,[JUMP (READ71)]       ; paire pointée simple ;

READ92:                                     ; cas :  . <s> ] ;
        XCONS   NIL,A1                      ; fabrique (<s>) ;
        SCDR    A1,A2                       ; qui devient le dernier élément ;
        POP     A1                          ; 1er doublet ;
        CAR     A1,A3                       ; pour tester la marque ;
        NEQ     A3,'*MARK*,[JUMP (ERLEC8)]  ; elle n'est pas marquée ;
        CDR     A1,A3                       ; A3 ← adresse 1er élément ;
        EQ      A3,A2,[JUMP (READ93)]       ; forme [ s1 . s2 ] => CONS ;
        SCAR    'MCONS,A1,[RETURN]          ; forme [s1 s2 ... sN-1 . sN] ;
READ93: SCAR    'CONS,A1,[RETURN]

        ; Erreurs à la lecture ;
        ;---------------------;

ERLEC1: MOVE    '1,A1,[JUMP (ERLEC)]
ERLEC3: MOVE    '3,A1,[JUMP (ERLEC)]
ERLEC4: MOVE    '4,A1,[JUMP (ERLEC)]
ERLEC6: MOVE    '6,A1,[JUMP (ERLEC)]
ERLEC7: MOVE    '7,A1,[JUMP (ERLEC)]
ERLEC8: MOVE    '8,A1,[JUMP (ERLEC)]

ERLEC:  PUSH    '*MARK*                     ; fin des arguments ;
        PUSH    '"Erreur de syntaxe :";     message ;
        PUSH    A1,,[JUMP (SERROR)]         ; numéro de l'erreur ;
```

```
            ; Fonctions standard ;
            ;--------------------;


            ; (READ) ;

            ENTRY   READ,0SUBR

READ:       NOP     ,,[CALL (RD1)]          ; première us ;
            EQ      A3,'1,[JUMP (READ)] ; pour sauter toutes les ) en trop ;
            NOP     ,,[JUMP (READ0)]        ; on lit vraiment ;

            ; (IMPLODE l) ;

            ENTRY   IMPLODE,1SUBR

IMPLODE:MOVE    A1,(@ IMPLD)            ; range la liste à interner ;
            MOVE    'T,(@ IMPLI),[CALL (READ)] ; indicateur IMPLODE=vrai ;
            MOVE    NIL,(@ IMPLI),[RETURN]; indicateur IMPLODE=faux ;

            ; (READCH) ;

            ENTRY   READCH,0SUBR

READCH: PUSH    (ASCIII),,[JUMP (GETCH)]

            ; (PEEKCH) ;

            ENTRY   PEEKCH,0SUBR

PEEKCH: NOP     ,,[CALL (GETCH)]
            MOVE    A4,(@ RINGUR),[JUMP (ASCIII)]

            ; (ASCII n) retourne le caractère de code n ;

            ENTRY   ASCII,1SUBR

ASCIII: MOVE    A4,A1                   ; ASCII interne ;
ASCII:  SINDEX  (BUFAT)                 ; prépare le tampon atome ;
            MOVEX   A1,'0                   ; charge le code du caractère ;
            MOVEX   '0,'1                   ; indique la fin du tanpon ;
            INTERN  (BUFAT),A1,[RETURN]     ; intern et rentre ;

            ; (CASCII c) retourne le code du caractère c ;

            ENTRY   CASCII,1SUBR

CASCII: PNAM    A1,(BUFCH)              ; représentation externe ;
            MOVE    (@ BUFCH),A1,[RETURN]   ; retourne le code et rentre ;

            ; (TYPECH c n) accès/modif au type du caractère ;

            ENTRY   TYPECH,2SUBR

TYPECH: NOP     ,,[CALL (CASCII)]       ; conversion du caractère ;
            SINDEX  (TABCH)                 ; adresse de la table ;
            TNIL    A2,,[JUMP (TYPECH1)]    ; lecture seule ;
            MOVEX   A2,A1                   ; force le nouveau type ;
TYPECH1:XMOVE   A1,A1,[RETURN]          ; le type actuel ;
```

```
                ; Données utilisées par le lecteur ;
                ;--------------------------------------;

RINGUR: DATA    0                       ; caractère à reingurgiter ;
RDPRD:  DATA    0                       ; prof. read (cf: pretty-read) ;

IMPLI:  DATA    NIL                     ; =NIL si READ, =T si IMPLODE ;
IMPLD:  DATA    NIL                     ; liste des caractères ;

BUFAT:  BLOCK   30,0                    ; tampon atome ;
BUFCH:  BLOCK   30,0                    ; tampon caractère ;

                ; Code des caractères de la table TABCH ;

                ;       type-ch = 0 :   caractères null (à ignorer) ;
                ;       type-ch = 1 :   début de commentaires  ;
                ;       type-ch = 2 :   fin de commentaires ;
                ;       type-ch = 3 :   quote caractère ;
                ;       type-ch = 4 :   début de liste ;
                ;       type-ch = 5 :   fin de liste ;
                ;       type-ch = 6 :   début de liste crochée ;
                ;       type-ch = 7 :   fin de liste crochée ;
                ;       type-ch = 8 :   point ;
                ;       type-ch = 9 :   séparateur nul ;
                ;       type-ch = 10 :  macro-caractère ;
                ;       type-ch = 11 :  délimiteur de chaîne de caractères ;
                ;       type-ch = 12 :  caractère normal ;

TABCH:                                  ; Table des caractères ;
        DATA    0                       ; 00 NULL ;
        BLOCK   7,12                    ; 01-07 ↑A-↑G ;
        BLOCK   5,9                     ; 10-14 ↑H-↑L BS TAB LF VT FF ;
        DATA    2                       ; 15 ↑M : RC ;
        BLOCK   2,12                    ; 16-17 ↑N-↑O ;
        BLOCK   8,12                    ; 20-27 ↑P-↑W ;
        BLOCK   8,12                    ; 30-37 ↑X-🄿 ;
        DATA    9                       ; 40 SP ;
        DATA    12                      ; 41 ! ;
        DATA    11                      ; 42 " ;
        DATA    12                      ; 43 # ;
        BLOCK   3,12                    ; 44-46 $ % & ;
        DATA    10                      ; 47 ' ;
        DATA    4                       ; 50 ( ;
        DATA    5                       ; 51 ) ;
        BLOCK   4,12                    ; 52-55 * + , - ;
        DATA    8                       ; 56 . ;
        DATA    3                       ; 57 / ;
        BLOCK   8,12                    ; 60-67 0 1 2 3 4 5 6 7 ;
        BLOCK   3,12                    ; 70-72 8 9 : ;
        DATA    1                       ; 73 point-virgule ;
        BLOCK   4,12                    ; 74-77 < = > ? ;
        BLOCK   8,12                    ; 100-107 @ A B C D E F G ;
        BLOCK   8,12                    ; 110-117 H I J K L M N O ;
        BLOCK   8,12                    ; 120-127 P Q R S T U V W ;
        BLOCK   3,12                    ; 130-132 X Y Z ;
        DATA    6                       ; 133 [ ;
        DATA    12                      ; 134 \ ;
        DATA    7                       ; 135 ] ;
        BLOCK   2,12                    ; 136-137 ↑ ← ;
        BLOCK   8,12                    ; 140-147 ç a b c d e f g ;
        BLOCK   8,12                    ; 150-157 h i j k l m n o ;
        BLOCK   8,12                    ; 160-167 p q r s t u v w ;
```

```
          BLOCK  7,12                    ; 170-172 x y z { | } ~ ;
          DATA   0                       ; 177 DEL ;
          ))

; Fin du lecteur ;

'OK

; Définition du macro-caractère d'activation ;

(DMC "ù" ()
   ; ù = ↑R ;
   [QUOTE (VCMC2
    ['(NOP NIL NIL CALL (READ)) '(NOP NIL NIL JUMP (PROBJT))])])

; Epilogue standard ;

(PROGN
   (ANACODE '(~reader))
   (STATUS 1 1)
   (PRINT "...  Lecteur VLISP - VCMC2 chargé.")
   (PRINT "     ùs pour (PRINT (READ))"))
```

CROSS REFERENCE

Signification des codes associés aux numéros des lignes :

définition de fonction de type DE DF DM DMC ou ENTRY
& définition de fonction de type ESCAPE, ESCLOOP
: définition d'étiquette dans PROG, DO, LAP ...
, variable argument d'une fonction
= variable affectée par SETQ ou SETQQ
' nom apparaissant dans une S-expression quotée
- code instruction assembleur

ù	DMC	475
[S	DMC	35
/'	ENTRY	208
ASCII	ENTRY	378
CASCII	ENTRY	388
CODE	DF	30
DMC	ENTRY	218
IMPLODE	ENTRY	357
PEEKCH	ENTRY	371
READ	ENTRY	349
READCH	ENTRY	365
TYPECH	ENTRY	395

1	/'	208#	212:									
2	*MARK*	247'	289'	297'	323'	339'						
3	0SUBR	208	349	365	371							
4	1SUBR	357	378	388								
5	2SUBR	395										
6	@	52	53	86	87	88	90	96	99	102	104	165
		168	169	170	200	359	360	361	374	391		
7	A1	173	185	188	189	189	203	213	214	221	223	256
		257	259	272	274	286	287	288	290	294	295	296
		298	303	304	316	319	320	321	322	324	326	327
		332	333	334	335	336	337	341	359	380	382	384
		390	391	400	401	401						
8	A2	174	178	182	183	184	187	193	195	196	197	202
		222	246	247	248	249	273	274	288	289	296	297
		305	316	320	325	399	400					
9	A3	94	124	125	128	129	131	133	134	135	152	175
		180	190	199	204	233	245	254	261	306	322	323
		324	325	352								
10	A4	63	87	94	96	97	98	99	100	100	101	102
		105	173	182	195	200	374	380				
11	ADD	165-	183-	196-								
12	ANACODE	483										
13	APPLY	174										
14	ASCII	173	378#	381:								
15	ASCIII	367	374	380:								
16	BLOCK	412-	413-	433-	434-	436-	437-	438-	443-	447-	450-	451-
		453-	454-	455-	456-	457-	461-	462-	463-	464-	465-	
17	BUFAT	181	186	188	194	201	203	381	384	412:		
18	BUFCH	101	102	390	391	413:						
19	CALL	91	123	127	130	151	173	174	179	198	212	220
		222	232	244	253	255	259	302	303	351	360	373

		397	478'									
20	CAR	100-	288-	296-	322-							
21	CASCII	388#	390:	397								
22	CDR	99-	324-									
23	CODE	30#	41									
24	CONS	214-	247-	327'								
25	DATA	154-	155-	156-	157-	158-	159-	160-	161-	162-	235-	236-
		237-	238-	239-	240-	263-	264-	265-	266-	267-	268-	308-
		309-	310-	311-	312-	313-	406-	407-	409-	410-	432-	435-
		439-	440-	441-	442-	444-	445-	446-	448-	449-	452-	458-
		459-	460-	466-								
26	DE	220										
27	DF	30										
28	DMC	35	218#	220:	475							
29	ENTRY	208	218	349	357	365	371	378	388	395		
30	EQ	86-	124-	128-	133-	134-	180-	199-	245-	254-	325-	352-
31	ERLEC	185	290	332	333	334	335	336	337	339:		
32	ERLEC1	98	332:									
33	ERLEC3	197	333:									
34	ERLEC4	236	238	239	334:							
35	ERLEC6	297	335:									
36	ERLEC7	308	310	312	313	336:						
37	ERLEC8	323	337:									
38	FALSE	245	254									
39	FLIST	98-										
40	FNIL	90-										
41	FSUBR	218										
42	GE	169-	197-									
43	GETCH	86:	123	127	130	179	367	373				
44	GETCH1	86	90:									
45	GETCH2	88	92:	102	105							
46	GETCH3	90	96:									
47	GETCH6	97	104:									
48	GETCV	124	126:	128	134	151	198					
49	GETCV1	123:	125	133								
50	GETCV2	129	133:									
51	IMPLD	96	99	104	359	410:						
52	IMPLI	53	90	360	361	409:						
53	IMPLODE	357#	359:									
54	IN	63-										
55	INCHB	63:	91									
56	INTERN	188-	203-	384-								
57	JUMP	86	88	90	97	98	102	105	124	125	128	129
		133	134	180	184	185	197	199	245	249	254	274
		278	282	290	297	316	323	325	332	333	334	335
		336	337	341	352	353	367	374	399	478'		
58	JUMPX	152-	233-	261-	306-							
59	L	30,	30									
60	LIST	298'										
61	LT	184-										
62	MCONS	326'										
63	MOVE	52-	53-	87-	88-	96-	102-	104-	105-	131-	170-	173-
		174-	175-	178-	185-	190-	193-	200-	204-	222-	246-	290-
		332-	333-	334-	335-	336-	337-	359-	360-	361-	374-	380-
		391-										
64	MOVEX	182-	187-	195-	202-	382-	383-	400-				
65	NEQ	125-	129-	289-	297-	323-						
66	NOP	35'	91-	123-	127-	130-	151-	179-	198-	212-	220-	232-
		244-	253-	255-	302-	351-	353-	373-	397-	478'	478'	
67	OK	471'										
68	OUTSTR	35										
69	PEEKCH	371#	373:									
70	PNAM	101-	390-									

71	POP	223-	273-	286-	287-	294-	295-	304-	305-	321-		
72	POPJ	156	157	158	240							
73	PRINT	485	486									
74	PROBJT	478'										
75	PROGN	482										
76	PUSH	221-	248-	249-	257-	259-	278-	282-	303-	339-	340-	341-
		367-										
77	QUOTE	214'	477									
78	RD1	151:	159	232	244	253	259	303	351			
79	RD2	162	192:									
80	RD21	194:	199									
81	RDMAC	160	172:									
82	RDPARF	155	167:									
83	RDPARO	154	164:									
84	RDPRD	165	168	169	170	407:						
85	RDSTR	161	177:									
86	RDSTR1	179:	184									
87	RDSTR2	180	186:									
88	RDTB1	152	154:									
89	READ	35	349#	351:	352	360	478'					
90	READ0	233:	255	353								
91	READ1	237	244:	282								
92	READ2	235	253:	278								
93	READ3	258:	274									
94	READ31	249	260:									
95	READ4	268	272:	278	282							
96	READ5	263	278:									
97	READ6	265	282:									
98	READ7	264	286:									
99	READ71	287:	316									
100	READ8	266	294:									
101	READ9	267	302:									
102	READ91	309	315:									
103	READ92	311	318:									
104	READ93	325	327:									
105	READCH	365#	367:									
106	READI	212	232:	302								
107	READINI	52:										
108	READT1	233	235:									
109	READT2	261	263:									
110	READT3	306	308:									
111	RETURN	53	63	94	131	135	165	169	170	175	190	204
		214	223	289	298	326	327	361	384	391	401	
112	RINGUR	52	86	87	88	200	374	406:				
113	RPLACA	40										
114	SCAR	298-	326-	327-								
115	SCDR	274-	316-	320-								
116	SCVAL	189-										
117	SERROR	341										
118	SINDEX	93-	181-	186-	194-	201-	381-	398-				
119	STATUS	26	484									
120	STRING	35										
121	SUB	135-	168-									
122	T	104'	360'									
123	TABCH	93	398	431:								
124	TNIL	97-	399-									
125	TYPECH	222	395#	397:								
126	TYPECH1	399	401:									
127	VCMC2	477										
128	XCONS	213-	256-	272-	319-							
129	XMOVE	94-	401-									
130	~reader	40'	483'									

APPENDICE D

```
;*********************************************;
; PRETTYPRINT (VLISP 8.2 & VLISP 11)     ;
;*********************************************;
;;
;-----    Fonctions internes ;

(DE P-P (l)
    ; Pretty-Print l'expression l  ;
    (COND
        ((NULL l) (PRINCH "(") (PRINCH ")"))
        ((ATOM l) (PRIN l))
        ((AND (EQ (CAR l) QUOTE) (NULL (CDDR l)))
            (PRINCH "'")
            (P-P (CADR L)))
        (T (PRINCH "(")
            (P-P (CAR l))
            (SELECTQ  (PTYPE (NEXTL l))
                (1 ; Format PROGN ; (P-PROGN))
                (2 ; Format WHILE ; (P-P1) (P-PROGN T))
                (3 ; Format DEF ; (P-P1) (P-P1) (P-PROGN T))
                (4 ; Format COND ; (P-COND))
                (5 ; Format SELECTQ ; (P-P1) (P-COND))
                (6 ; Format SETQ multiple ;
                    (T+3)
                    (WHILE l (P-P1) (P-P1) (IF l (TERPRI)))
                    (T-3))
                (; Format standard ;
                    (T+3)
                    (WHILE (LISTP l) (P-P1))
                    (T-3)))
            (AND l (PRINCH " ") (PRINCH ".") (PRINCH " ") (PRIN l))
            (PRINCH ")"))))

 (DE P-P1 ()
    ; Pretty Print un élément de la liste l  ;
    ; mais pas le 1er ! ;
    (PRINCH " ")
    (P-P (NEXTL l)))

 (DE P-PROGN (?)
    ; Pretty Print le PROGN l si ?=T il indente toujours ;
    (IF (AND (NULL (CDR l)) (NULL ?))
        ; 1 seul argument ;
        (P-P1)
```

```
        ; plusieurs arguments ;
        (T+3)
        (WHILE (LISTP l)
           (IF (GT (LMARGIN) (OUTPOS)) (OUTPOS (LMARGIN)) (TERPRI))
           (P-P (NEXTL l)))
        (T-3)))

(DE P-CONDS ()
   ; Pretty Print le COND l  ;
   (T+3)
   (WHILE (LISTP l)
      (TERPRI)
      (PRINCH "(")
      (LET (l (NEXTL l)) (P-P (NEXTL l)) (IF l (P-PROGN)))
      (PRINCH ")"))
   (T-3))

(DE T+3 ()
   ; effectue un renfoncement droit ;
   (LMARGIN (+ (LMARGIN) 3)))

(DE T-3 ()
   ; enlève un renfoncement droit ;
   (LMARGIN (- (LMARGIN) 3)))

;*****    Fonctions standard du PRETTY PRINT ;

(DF PRETTY (l ;; x)
   ; Pretty Print la fonction (CAR l) ;
   (STATUS PRINT 7)
   (LMARGIN 0)
   (SETQ
      x
        (CDR
          (ASSQ (TYPEFN (CAR l)) '((EXPR . DE) (FEXPR . DF) (MACRO . DM)))))
   (IF x (P-P (MCONS x (CAR l) (GETFN (CAR l)))))
   (TERPRI)
   (STATUS PRINT 0)
   (CAR l))

(DF PRETTYF (l)
   ; Pretty Print le fichier (CAR l) ;
   (INPUT (CAR l))
   (OUTPUT (CAR l))
   (STATUS PRINT 7)
   (LMARGIN 0)
   (ESCAPE EOF (WHILE T (P-P (READ)) (TERPRI 2)))
   (OUTPUT)
   (INPUT)
   (STATUS PRINT 0)
   (CAR l))

(DF SAVEF (l)
   ; Sauve des définitions de fonctions Pretty Printées ;
   ; (SAVEF file fnt1 ... fntN) ;
   (OUTPUT (NEXTL l))
   (WHILE (LISTP l) (EVAL (LIST 'PRETTY (NEXTL L))) (TERPRI))
   (OUTPUT))

;*****    Mise en place des F-TYP des atomes standard ;

(PROGN
   (ESCAPE EOF (WHILE T (PTYPE (READ) (READ))))
```

```
      (INPUT)
      (PRINT "Pretty charge : PRETTY SAVEF"))


;;;;

; Type 1 : Format PROGN ;

AND      1
EXIT     1
LIST     1
MCONS    1
OR       1
PRIN     1
PRINT    1
PROG1    1
PROGN    1

; Type 2 : Format LAMBDA ;

ESCAPE   2
IF       2
IFN      2
LAMBDA   2
LET      2
MAP      2
MAPC     2
MAPCAR   2
MAPCONC  2
MAPLIST  2
UNTIL    2
WHERE    2
WHILE    2

; Type 3 : Format DEF ;

DE                 3
DF                 3
DM                 3
DMC                3

; Type 4 : Format COND ;

COND               4

; Type 5 : Format SELECTQ ;

SELECTQ            5

; Type 6 : Format SETQ multiple ;

SETQ               6

; Fin des P-TYP ;

()                 ()
'/
```

CROSS REFERENCE

Signification des codes associés aux numéros des lignes :

#	définition de fonction de type DE DF DM DMC ou ENTRY
&	définition de fonction de type ESCAPE, ESCLOOP
:	définition d'étiquette dans PROG, DO, LAP ...
,	variable argument d'une fonction
=	variable affectée par SETQ ou SETQQ
'	nom apparaissant dans une S-expression quotée
-	code instruction assembleur

P-COND	DE	52
P-P	DE	7
P-P1	DE	34
P-PROGN	DE	40
PRETTY	DF	72
PRETTYF	DF	85
SAVEF	DF	97
T+3	DE	62
T-3	DE	66

1	+	64								
2	-	68								
3	?	40,	42							
4	AND	12	31	42						
5	ASSQ	79								
6	ATOM	11								
7	CADR	14								
8	CAR	12	16	79	80	80	83	87	88	95
9	CDDR	12								
10	CDR	42	78							
11	COND	9								
12	DE	7	34	40	52	62	66	79'		
13	DF	72	79'	85	97					
14	DM	79'								
15	EOF	91&	107&							
16	EQ	12								
17	ESCAPE	91	107							
18	EVAL	101								
19	EXPR	79'								
20	FEXPR	79'								
21	GETFN	80								
22	GT	48								
23	IF	25	42	48	58	80				
24	INPUT	87	93	108						
25	L	14	49	101						
26	LET	58								
27	LIST	101								
28	LISTP	29	47	55	101					
29	LMARGIN	48	48	64	64	68	68	75	90	
30	MACRO	79'								
31	MCONS	80								
32	NEXTL	17	38	49	58	58	100	101		
33	NULL	10	12	42	42					
34	OUTPOS	48	48							

35	OUTPUT	88	92	100	102							
36	P-COND	21	22	52#								
37	P-P	7#	14	16	38	49	58	80	91			
38	P-P1	19	20	20	22	25	25	29	34#	44		
39	P-PROGN	18	19	20	40#	58						
40	PRETTY	72#	101'									
41	PRETTYF	85#										
42	PRIN	11	31									
43	PRINCH	10	10	13	15	31	31	31	32	37	57	59
44	PRINT	74	82	89	94	109						
45	PROGN	106										
46	PTYPE	17	107									
47	QUOTE	12										
48	READ	91	107	107								
49	SAVEF	97#										
50	SELECTQ	17										
51	SETQ	76										
52	STATUS	74	82	89	94							
53	T	15	19	20	91	107						
54	T+3	24	28	46	54	62#						
55	T-3	26	30	50	60	66#						
56	TERPRI	25	48	56	81	91	101					
57	TYPEFN	79										
58	WHILE	25	29	47	55	91	101	107				
59	(	7,	10	11	11	12	12	16	17	25	25	29
		31	31	38	42	47	55	58	58	58	58	72,
		79	80	80	83	85,	87	88	95	97,	100	101
60	×	72,	77=	80	80							

APPENDICE E

```
;                    V C M C 2 P   .   V L I                        ;
;                                                                    ;
;              IMPRIMEUR VLISP en machine VCMC2                      ;
;--------------------------------------------------------------------;
;          Ce fichier doit être édité au moyen de :                  ;
;                (CROSSF V2P T T T {NIL/T} T T)                      ;
;                   {NIL/T} si VERSATEC                              ;
;--------------------------------------------------------------------;
;                                                                    ;
;                    Jérôme CHAILLOUX                                ;
;                                                                    ;
;          Département d'Informatique                                ;
;          Université de Paris 8 - Vincennes                         ;
;          Route de la Tourelle 75571 Paris Cédex 12                 ;
;          Tél : 374 12 50 poste 299                                 ;
;                                                                    ;
;          L.I.T.P.   (CNRS LA 248)                                  ;
;          2 Place Jussieu 75221 Paris Cédex 05                      ;
;          Tél : 336 25 25 poste 53-70                               ;
;                                                                    ;
;          I.R.C.A.M.                                                ;
;          31 Rue St Merri 75004 Paris                               ;
;          Tél : 277 12 33 poste 48-48                               ;
;--------------------------------------------------------------------;

(STATUS 2 1 2)

; Pour prettyprinter correctement le code ;

(DF CODE (L) L)


; Pour contrôler la lecture ;

(DMC "S" () (PRIN1 (READ)) '(NOP))
```

```
; Imprimeur VLISP ;

(RPLACA '~pprinter
 (CODE NIL

          ;┌──────────────────────────────────────────────────┐;
          ;│                                                  │;
          ;│            Impressions VLISP - VCMC2             │;
          ;│            Printer et Pretty-printer             │;
          ;│                                                  │;
          ;└──────────────────────────────────────────────────┘;


          ; Routine d'initialisation de l'imprimeur ;
          ; à utiliser en cas d'erreur pour reinitialiser ;
          ; l'imprimeur ;

PRININI:MOVE     NIL,(@ IEXPLD)          ; raz indicateur EXPLODE ;
        MOVE     '100,(@ PRMDP)          ; init profondeur max du PRINT ;
        MOVE     '1000,(@ PRMLN)         ; init longeur max du PRINT ;
        MOVE     '0,(@ POCOUR)           ; position début de ligne ;
        MOVE     '0,(@ NBLEFT)           ; init marge gauche ;
        MOVE     '70,(@ NBRIG)           ; init marge droite ;
        SINDEX   (BUFOUT)                ; pour nettoyer  BUFOUT ;
        MOVE     '0,A1                   ; index ;
        SILENCE                          ; la boucle est trop longue ... ;
PRII1:  MOVEX    '" ",A1                 ; force un espace ;
        ADD      '1,A1                   ; incrémente l'index ;
        LT       A1,'132,[JUMP (PRII1)]  ; pour les 132 positions ;
        REVIVE                           ; restaure les traces ;
        NOP      ,,[RETURN]              ; et c'est tout ;

        ; FULLIN : appelle l'IT Soft : EOL si la ligne déborde ;

FULLIN: PUSH     A1                      ; sauve tous les registres ;
        PUSH     A2                      ; encore ;
        PUSH     A3                      ; encore ;
        PUSH     A4                      ; et encore ;
        MOVE     'EOL,A1                 ; nom de l'IT Soft ;
        MOVE     NIL,A2,[CALL (ITSOFT)]  ; sans argument ;
        POP      A4                      ; restaure les registres ;
        POP      A3                      ; encore ;
        POP      A2                      ; encore ;
        POP      A1,,[RETURN]            ; et encore ;

        ; (EOL) 0SUBR  équivalent à (TERPRI) ;
        ; mais est appelée par IT Soft ;

        ENTRY    EOL,0SUBR

EOL:    NOP      ,,[CALL (OUTLIN)]       ; vide la ligne ;
        MOVE     NIL,A1,[RETURN]         ; et retourne A1 ;
```

```
            ; Routines de base ;
            ;------------------;

            ; OUTLIN : vide le tampon de sortie ;
            ; et change de ligne ;

OUTLIN: FNIL    (@ IEXPLD),,[JUMP (EXPLS)] ; C'est la fonction EXPLODE ;
        SINDEX (BUFOUT)                   ; init base index ;
        MOVE    '0,A2,[JUMP (OUTLI2)]     ; index sur BUFOUT ;
OUTLI1: XMOVE   A2,A1                     ; caractère suivant ;
        OUT     A1                        ; qui est imprimé ;
        MOVEX   '" ",A2                   ; remet à espace le tampon ;
        ADD     '1,A2                     ; incrémente l'index ;
        SUB     '1,(@ POCOUR)             ; décompte des caractères ;
OUTLI2: GT      (@ POCOUR),'0,[JUMP (OUTLI1)] ; il en reste ;
        OUT     '13                       ; imprime 'RC' ;
        OUT     '10                       ; imprime 'LF' ;
        MOVE    (@ NBLEFT),(@ POCOUR)     ; init pointeur courant ;
        ADD     '1,(@ PRCLP)              ; compteur nb de lignes max ;
        NEQ     (@ PRCLP),(@ PRMLP),[RETURN] ; tout est normal ;
        SSTACK  (@ SPRINT),,[JUMP (PROBJE)] ; retour très rapide ;

            ; OUTCH : suppose dans A4 un caractère, le depose dans ;
            ; le tampon de sortie  et actualise le pointeur courant ;

OUTCH:  FNIL    (@ IEXPLD),,[JUMP (EXPLCH)] ; C'est la fonction EXPLODE ;
        GE      (@ POCOUR),(@ NBRIG),[JUMP (FULLIN)] ; ligne pleine ;
        SINDEX (BUFOUT)                   ; init base index ;
        MOVEX   A4,(@ POCOUR)             ; charge le caractère ;
        ADD     '1,(@ POCOUR),[RETURN]; incrémente le pointeur courant ;

            ; OUTSP : édite un espace si c'est possible ;
            ; sinon change de ligne ;

OUTSP:  FNIL    (@ IEXPLD),,[JUMP (EXPLS)] ; C'est la fonction EXPLODE ;
        GE      (@ POCOUR),(@ NBRIG),[JUMP (FULLIN)] ; line pleine ;
        SINDEX (BUFOUT)                   ; init base index ;
        MOVEX   '" ",(@ POCOUR)           ; charge l'espace ;
        ADD     '1,(@ POCOUR),[RETURN]; actualise le pointeur courant ;

            ; PRATOM : édite l'atome dans A1 ;
            ; change de ligne si l'atome ne rentre pas dans la ligne ;

PRATOM: PLEN    A1,A4                     ; A4 ← longeur du P-name ;
        ADD     (@ POCOUR),A4             ; calcul la nouvelle taille ;
        GT      A4,(@ NBRIG),[CALL (FULLIN)] ; ca rentre pas ;
        PNAM    A1,(BUFAT)                ; BUFAT ← les caractères ;
        MOVE    '0,A2                     ; index sur BUFAT ;
        PLEN    A1,A3,[JUMP (PRAT3)]      ; compteur ;
PRAT2:  SINDEX (BUFAT)                    ; charge l'index ;
        XMOVE   A2,A4,[CALL (OUTCH)]      ; édite 1 caractère ;
        ADD     '1,A2                     ; avance l'index sur PNAM ;
PRAT3:  SUB     '1,A3                     ; compte ;
        GE      A3,'0,[JUMP (PRAT2)]      ; il en reste ;
        NOP     ,,[RETURN]                ; c'est fini ;
```

```
            ; Impression standard ;
            ;-----------------------;

            ; PROBJ : fonction  d'impression interne de A1 ;

PROBJ:      PUSH    A1                      ; sauve la valeur à éditer ;
            STACK   (@ SPRINT)              ; pour un retour rapide ;
            MOVE    '0,(@ PRCLP)            ; init nb delignes ;
            MOVE    '0,(@ PRCLN)            ; init la longeur courante ;
            MOVE    '0,(@ PRCDP),[CALL (PROBJ0)] ; init la prof courante ;
PROBJE:                                     ; en cas de retour subit ;
            POP     A1,,[RETURN]            ; ramène en valeur l'objet édité ;

PROBJ0:     FLIST   A1,,[JUMP (PRATOM)]     ; c'est tout pour les atomes ;
            ADD     '1,(@ PRCDP)            ; actualise la prof courante ;
            LE      (@ PRCDP),(@ PRMDP),[JUMP (PROBJ2)] ; ca tient ;
            MOVE    '"&",A4,[JUMP (OUTCH)]; "&" suffit dans ce cas ;

PROBJ2:     CAR     A1,A2              ; récupère le 1er élément de la liste ;
            NEQ     A2,'QUOTE,[JUMP (PROBJ5)] ; c'est pas ' ;
            CDR     A1,A2                   ; (larg) ;
            CDR     A2,A3
            FNIL    A3,,[JUMP (PROBJ5)]     ; Ce n'est pas la fonction QUOTE ;
            MOVE    '"'",A4,[CALL (OUTCH)]; imprime le caractère ' ;
            CAR     A2,A1,[JUMP (PROBJ0)]   ; et l'argument de la fonction ;

PROBJ5:     MOVE    '"(",A4,[CALL (OUTCH)]; début de liste ;
            NOP     ,,[JUMP (PROBJ7)]

PROBJ6:     NOP     ,,[CALL (OUTSP)]        ; séparateur d'élément de liste ;
PROBJ7:     ADD     '1,(@ PRCLN)            ; actualise la longeur courante ;
            LE      (@ PRCLN),(@ PRMLN),[JUMP (PROBJ8)] ; c'est bon ;
            PUSH    (PROBJ9),,[CALL (PROBJD)]
                                            ; édite 3 points de suspension ;
            NOP     ,,[CALL (PROBJD)]       ; la pile compte en binaire ;
PROBJD:     MOVE    '".",A4,[JUMP (OUTCH)]; utilise le JRST hack ;

PROBJ8:     CDR     A1,TST                  ; sauve le reste de la liste ;
            CAR     A1,A1,[CALL (PROBJ0)]   ; édite l'élément suivant ;
            POP     A1                      ; récupère le reste ;
            TLIST   A1,,[JUMP (PROBJ6)]     ; la liste continue ;
            TNIL    A1,,[JUMP (PROBJ9)]     ; c'est une belle fin ;
            NOP     ,,[CALL (OUTSP)]
            MOVE    '".",A4,[CALL (OUTCH)]
            NOP     ,,[CALL (OUTSP)]
            NOP     ,,[CALL (PRATOM)]       ; édite l'élément pointé ;
PROBJ9:     MOVE    '")",A4,[CALL (OUTCH)]; fin de la liste ;
            SUB     '1,(@ PRCDP),[RETURN]   ; actualise profondeur courante ;

            ; PROBJT : imprime A1 et change de ligne ;

PROBJT:     PUSH    (FULLIN),,[JUMP (PROBJ)] ; prépare le retour ;
```

```
        ; Fonctions de sortie standard ;
        ;----------------------------;

        ; (PRIN e1 ... eN)   FSUBR ;

        ENTRY   PRIN,FSUBR

PRIN11: CDR     A1,TST,[CALL (EVCAR)]
        NOP     ,,[CALL (OUTSP)]
        NOP     ,,[CALL (PROBJ)]
        MOVE    A1,A2
        POP     A1
PRIN:   TLIST   A1,,[JUMP (PRIN11)]
        MOVE    A2,A1,[RETURN]

        ; (PRINT e1 ... eN)   FSUBR ;

        ENTRY   PRINT,FSUBR

PRINT:  NOP     ,,[CALL (PRIN)]
        NOP     ,,[JUMP (FULLIN)]

        ; (TERPRI n)   1SUBR ;

        ENTRY   TERPRI,1SUBR

TERPRI: TNUMB   A1,,[JUMP (TERPR2)]       ; c'est un bon argument ;
        MOVE    '1,A1                     ; n = 1 par défaut ;
TERPR1: SUB     '1,A1,[CALL (OUTLIN)]     ; compte ;
TERPR2: GT      A1,'0,[JUMP (TERPR1)]     ; il en faut encore ;
        NOP     ,,[RETURN]                ; et retourne 0 ;

        ; (PRINCH c n)   2SUBR ;

        ENTRY   PRINCH,2SUBR

PRINCH: MOVE    A1,A4                     ; pour OUTCH ;
        TNUMB   A2,,[JUMP (PRINC2)]       ; le 2ème argument est correct ;
        MOVE    '1,A2,[JUMP (PRINC2)]     ; 1 par défaut ;
PRINC1: PUSH    A2,,[CALL (OUTCH)]        ; envoyer ;
        POP     A2
        SUB     '1,A2                     ; décompte ;
PRINC2: GT      A2,'0,[JUMP (PRINC1)]     ; il en reste ;
        NOP     ,,[RETURN]                ; retourne le caractère ;
```

```
              ; Routines internes du PRETTY-PRINT ;
              ;--------------------------------------;

              ; PP  : pretty-prin A1 ;

      PPCAR:                                    ; (PP (CAR A1)) ;
              CAR      A1,A1
      PP:                                       ; (PP A1) ;
              FNIL     A1,,[JUMP (PP10)]        ; ce n'est pas NIL ;
              MOVE     '"(",A4,[CALL (OUTCH)]
              MOVE     '")",A4,[JUMP (OUTCH)]; NIL s'imprime toujours () ;
      PP10:   FLIST    A1,,[JUMP (PRATOM)]      ; impression de l'atome simple ;
              PUSH     A1                       ; sauve la liste ;
              CAR      A1,A1                    ; A1 ← la fonction ;
              MOVE     'PRETTY,A2,[CALL (GET)]
                                   ; recherche de format utilisateur (DMP) ;
              TNIL     A1,,[JUMP (PP11)]        ; yen a pas : formats standard ;
              XTOPST   A1,,[JUMP (EVFEXP)]      ; appel de la fonction user ;
      PP11:   POP      A1                       ; récupère  la liste à imprimer ;
              CAR      A1,A2                    ; 1er element ;
              CDR      A1,A3                    ; reste de la liste ;
              NEQ      A2,'QUOTE,[JUMP (PP12)]
                                                ; ce n'est pas la fonction QUOTE ;
              CDR      A3,A4                    ; 2ème argument ;
              FNIL     A4,,[JUMP (PP12)]        ; ce n'est pas la fonction QUOTE ;
              MOVE     '"'",A4,[CALL (OUTCH)]; impression sous forme 's ;
              CAR      A3,A1,[JUMP (PP)]        ; du seul argument ;
      PP12:   MOVE     '"(",A4,[CALL (OUTCH)]
              CDR      A1,TST                   ; sauve les arguments ;
              CAR      A1,TST,[CALL (PPCAR)]    ; imprime le nom de la fonction ;
              PTYP     TST,A2                   ; A2 ← Printype de la fonction ;
              POP      A1                ; A1 ← les arguments de la fonction ;
              PUSH     (PPTE)            ; prépare le retour des impressions ;
              JUMPX    (PPTBLX),A2          ; branchement indexé sur le P-TYP ;

      PPTBLX: DATA     (PPT0)                   ; type 0 : inconnu ;
              DATA     (PPT1)                   ; type 1 : DE ;
              DATA     (PPT2)                   ; type 2 : IF ;
              DATA     (PPT3)                   ; type 3 : PROGN ;
              DATA     (PPT4)                   ; type 4 : SELECTQ ;
              DATA     (PPT5)                   ; type 5 : COND ;
              DATA     (PPT6)                   ; type 6 : SETQ ;

              ; Retour unique pour tout type ;

      PPTE:   TNIL     A1,,[JUMP (PPTE7)]       ; c'est une vraie fin ;
              NOP      ,,[CALL (OUTSP)]
              MOVE     '".",A4,[CALL (OUTCH)]; symbole de paire pointée ;
              NOP      ,,[CALL (OUTSP)]
              NOP      ,,[CALL (PP)]            ; imprime la partie droite ;
      PPTE7:  MOVE     '")",A4,[JUMP (OUTCH)]; et ferme la liste ) ;

              ; Imprime l'élément suivant de A1 ;
              ; ce n'est pas le premier ;

      PP1:    NOP      ,,[CALL (OUTSP)]         ; séparateur d'éléments ;
              CDR      A1,TST,[CALL (PPCAR)]    ; imprime l'élément ;
              POP      A1,,[RETURN]             ; restaure de CDR ;
```

```
            ; Différents formats du PP ;
            ;————————————————————————————;


            ; Type 0 : PTYP indéterminé ;

PPT0:       ADD     '3,(@ NBLEFT),[JUMP (PPT03)] ; prépare un renfoncement ;
PPT02:      NOP     ,,[CALL (PP1)]        ; imprime l'élément suivant ;
PPT03:      TLIST   A1,,[JUMP (PPT02)]    ; il reste des éléments ;
            SUB     '3,(@ NBLEFT),[RETURN]; et enlève le renfoncement ;

            ; Type 1 : PTYPE de type DE ;

PPT1:       NOP     ,,[CALL (PP1)]     ; élément suivant sur la même ligne ;
                                          ; Type IF doit suivre ... ;
            ; Type 2 : PTYPE de type IF ;

PPT2:       NOP     ,,[CALL (PP1)]     ; élément suivant sur la même ligne ;
            NOP     ,,[JUMP (PPT31)]      ; vers le PROGN véritable ;

            ; Type 3 : PTYPE de type PROGN ;

PPT3:       CDR     A1,A2
            TNIL    A2,,[CALL (PP1)]      ; s'il n'y a qu'un élément ;
PPT31:      ADD     '3,(@ NBLEFT),[JUMP (PPT33)] ; prépare un renfoncement ;
PPT32:      NOP     ,,[CALL (FULLIN)]     ; change de ligne ;
            NOP     ,,[CALL (PP1)]        ; imprime l'élément suivant ;
PPT33:      TLIST   A1,,[JUMP (PPT32)]    ; il reste des éléments ;
            SUB     '3,(@ NBLEFT),[RETURN]; et enlève le renfoncement ;

            ; Type 4 : PTYPE de type SELECTQ ;

PPT4:       NOP     ,,[CALL (PP1)]        ; sélecteur sur la même ligne ;
                                          ; Type COND doit suivre ... ;
            ; Type 5 : PTYPE de type COND ;

PPT5:       ADD     '3,(@ NBLEFT),[JUMP (PPT55)] ; renfoncement des clauses ;
PPT51:      NOP     ,,[CALL (FULLIN)]     ; une nouvelle ligne par clause ;
            MOVE    '"(",A4,[CALL (OUTCH)]
            CDR     A1,TST                ; sauve le reste des clauses ;
            CAR     A1,A1                 ; A1 ← la clause ;
            CDR     A1,TST,[CALL (PPCAR)] ; sauve le corps de la clause ;
            POP     A1                    ; récupère le corps de la clause ;
            TLIST   A1,,[CALL (PPT3)]     ; corps édité comme un PROGN ;
            MOVE    '")",A4,[CALL (OUTCH)]; ferme la clause ;
            POP     A1                    ; le reste des clauses ;
PPT55:      TLIST   A1,,[JUMP (PPT51)]    ; il en reste ;
            SUB     '3,(@ NBLEFT),[RETURN]; retire le renfoncement ;

            ; Type 6 : PTYPE de type SETQ ;

PPT6:       FLIST   A1,,[RETURN]          ; c'est (SETQ) ;
            ADD     '3,(@ NBLEFT)         ; prépare un renfoncement ;
PPT61:      CDR     A1,TST,[CALL (PPCAR)] ; pp la variable ;
            POP     A1                    ; récupère le reste ;
            CDR     A1,TST,[CALL (PPCAR)] ; pp la valeur ;
            POP     A1                    ; récupère le couple suivant ;
            TLIST   A1,,[CALL (FULLIN)]   ; change de ligne ;
            TLIST   A1,,[JUMP (PPT61)]    ; ca continue ;
            SUB     '3,(@ NBLEFT),[RETURN]; enlève le renfoncement ;
```

```
              ; Fontions standard du PRETTY-PRINT ;
              ;-----------------------------------;

              ; (PPRIN s) Pretty-prin l'expression s ;
              ; (PPRINT s) Pretty-print l'expression s ;
              ; (PRETTY at) Pretty-print la fonction de at ;

              ENTRY   PPRIN,1SUBR
              ENTRY   PPRINT,1SUBR
              ENTRY   PRETTY,FSUBR

              ; PPRIN : retourne l'argument ;

     PPRIN:   PUSH    A1,,[CALL (PP)]          ; sauve l'argument et le PP ;
              POP     A1,,[RETURN]             ; retourne l'argument ;

              ; PPRINT : retourne l'argument ;

     PPRINT:  PUSH    A1,,[CALL (PP)]          ; sauve l'argument et le PP ;
              POP     A1,,[JUMP (FULLIN)]      ; change de ligne ;

              ; PRETTY : retourne NIL s'il n'y a rien d'imprimé ;

     PRETTY:  CAR     A1,A2                    ; A2 ← le nom de l'atome ;
              FATOM   A2,,[JUMP (FALSE)]       ; ce n'est pas un nom ;
              FVAL    A2,A1                    ; A1 ← la F-VAL ;
              TNIL    A1,,[JUMP (FALSE)]       ; rien a faire ;
              CONS    A2,A1                    ; fabrique (nom . fval) ;
              FTYP    A2,A3                    ; recupère le F-TYP ;
              NEQ     A3,'7,[JUMP (PRETT3)]    ; ce n'est pas EXPR ;
              CONS    'DE,A1,[JUMP (PPRINT)]   ; A1 ← (DE nom . fval) ;
     PRETT3:  NEQ     A3,'8,[JUMP (PRETT4)]    ; ce n'est pas FEXPR ;
              CONS    'DF,A1,[JUMP (PPRINT)]   ; A1 ← (DF nom . fval) ;
     PRETT4:  NEQ     A3,'9,[JUMP (PRETT6)]    ; ce n'est pas MACRO ;
              CONS    'DM,A1,[JUMP (PPRINT)]   ; A1 ← (DM nom . fval) ;
     PRETT6:  PUSH    A2                       ; sauve le nom ;
              MOVE    A2,A1                    ; A1 ← nom de l'atome ;
              MOVE    'PRETTY,A2,[CALL (GET)]  ; recherche de fonction DP ;
              POP     A2                       ; récupère le nom ;
              TNIL    A1,,[JUMP (FALSE)]       ; ya vraiment pas de définition ;
              CONS    A2,A1                    ; Fabrique (nom (larg) . body) ;
              CONS    'DMP,A1,[JUMP (PPRINT)]  ; A1 ← (DMP nom . fval ) ;

              ; (DMP nom (larg) . corps ) ;
              ; Définition de format d'édition utilisateur ;

              ENTRY   DMP,FSUBR

     DP:      CDR     A1,A2                    ; A2 ← ((larg) . corps) ;
              CAR     A1,A1                    ; A1 ← le nom de la fonction ;
              MOVE    'PRETTY,A3,[JUMP (PUT)]  ; et crée la fonction ;
```

```
          ; Autres fonctions standard de sortie ;
          ;-----------------------------------;

          ; (PLENGTH at) ramène le nombre de caractères ;
          ; du P-NAME de l'atome at ;

          ENTRY   PLENGTH,1SUBR

PLENGTH:PLEN    A1,A1,[RETURN]          ; et c'est tout ;

          ; (PTYPE at n)   2SUBR ;

          ENTRY   PTYPE,2SUBR

PTYPE:  TNIL    A2,,[JUMP (PTYP1)]      ; pas de 2ème arg ;
        SPTYP   A2,A1                   ; force le nouveau PTYP ;
PTYP1:  PTYP    A1,A1,[RETURN]          ; retourne le PTYP courant ;

          ; (PRINTLINE n)   régle le nb de lignes maximum d'impression ;

          ENTRY   PRINTLINE,1SUBR

PRINTLINE:
        TNIL    A1,,[JUMP (PRINTN1)]    ; il n'y a pas d'argument ;
        MOVE    A1,(@ PRMLP)            ; change la profondeur maximum ;
PRINTN1:MOVE    (@ PRMLP),A1,[RETURN]
                              ; retourne la profondeur maximum courante ;

          ; (PRINTLEVEL n)   régle la profondeur maximum d'impression ;

          ENTRY   PRINTLEVEL,1SUBR

PRINTLEVEL:
        TNIL    A1,,[JUMP (PRINTL1)]    ; il n'y a pas d'argument ;
        MOVE    A1,(@ PRMDP)            ; change la profondeur maximum ;
PRINTL1:MOVE    (@ PRMDP),A1,[RETURN]
                              ; retourne la profondeur maximum courante ;

          ; (PRINTLENGTH n)   régle la longeur maximum d'impression ;

          ENTRY   PRINTLENGTH,1SUBR

PRINTLENGTH:
        TNIL    A1,,[JUMP (PRINTN1)]    ; il n'y a pas d'argument ;
        MOVE    A1,(@ PRMLN)            ; change la longeur maximum ;
PRINTN1:MOVE    (@ PRMLN),A1,[RETURN]
                              ; retourne la longeur maximum courante ;
```

```
            ; Fonctions sur le tampon de sortie ;
            ;----------------------------------;

            ; S.P. TESPOS : teste si la position A1 reste bien ;
            ;          à l'intérieur de la ligne ;
TESPOS:  FNUMB   A1,,[JUMP (POSER)]        ; ce doit être un nombre ;
         LT      A1,'0,[JUMP (POSER)]      ; qui plus est positif ;
         LE      A1,(@ NBRIG),[RETURN]     ; et plus petit que NBRIG ;
                                           ; POSER doit suivre ... ;
POSER:   PUSH    '*MARK*                   ; marque fin arguments ;
         PUSH    '"Débordement de ligne :"
         PUSH    A1,,[JUMP (SERROR)]       ; argument erronné ;

            ; (LMARGIN n) régle la marge gauche ;

         ENTRY   LMARGIN,1SUBR

LMARGIN: TNIL    A1,,[JUMP (LMARG1)]       ; il n'y a pas d'argument ;
         NOP     ,,[CALL (TESPOS)]         ; teste la validité de l'argument ;
         MOVE    A1,(@ NBLEFT)      ; change la taille de la marge gauche ;
LMARG1:  MOVE    (@ NBLEFT),A1,[RETURN]
                                    ; retourne la marge gauche courante ;

            ; (RMARGIN n) régle la marge droite ;

         ENTRY   RMARGIN,1SUBR

RMARGIN: TNIL    A1,,[JUMP (RMARG1)]       ; il n'y a pas d'argument ;
         FNUMB   A1,,[JUMP (POSER)]        ; ce doit être un nombre ;
         LT      A1,'0,[JUMP (POSER)]      ; qui plus est positif ;
         GT      A1,'132,[JUMP (POSER)]    ; et plus petit que 132 ;
         MOVE    A1,(@ NBRIG)       ; change la taille de la marge droite ;
RMARG1:  MOVE    (@ NBRIG),A1,[RETURN]
                                    ; retourne la marge droite courante ;

            ; (OUTPOS n) régle la position courante ;

         ENTRY   OUTPOS,1SUBR

OUTPOS:  TNIL    A1,,[JUMP (OUTPO1)]       ; il n'y a pas d'argument ;
         NOP     ,,[CALL (TESPOS)]         ; teste la validité de l'argument ;
         MOVE    A1,(@ POCOUR)             ; change la position courante ;
OUTPO1:  MOVE    (@ POCOUR),A1,[RETURN]    ; retourne la position courante ;

            ; (OUTBUF n c)   charge directement le tampon ;

         ENTRY   OUTBUF,2SUBR

OUTBUF:  SINDEX  (OUTBUF),,[CALL (TESPOS)]
                                    ; init base index et teste l'argument ;
         TNIL    A2,,[JUMP (OUTBU1)]       ; il n'y a pas de 2ème argument ;
         MOVEX   A2,A1                     ; charge le tampon ;
OUTBU1:  XMOVE   A1,A1,[RETURN]            ; retourne le caractère présent ;
```

```
            ; Fonction EXPLODE ;
            ;-------------------;

            ; Utilise les mêmes routines que les ;
            ; fonctions standard de sortie ;

            ENTRY   EXPLODE,1SUBR

EXPLODE:MOVE    NIL,A2                  ; prépare la liste résultat ;
        XCONS   NIL,A2                  ; fabrique le 1er doublet ;
        PUSH    A2                      ; sauve pour le GC et le retour ;
        MOVE    A2,(@ LEXPLD)           ; liste courante en formation ;
        MOVE    'T,(@ IEXPLD)           ; indicateur EXPLODE = vrai ;
        NOP     ,,[CALL (PROBJ)]        ; explosion de l'objet ;
        MOVE    NIL,(@ IEXPLD)          ; indicateur EXPLODE = faux ;
        CDR     TST,A1,[RETURN]        ; retourne la liste des caractères ;

            ; Rajoute un espace à la liste des caractères ;

EXPLS:  PUSH    A1                      ; Registres de travail ;
        PUSH    A2
        MOVE    '" ",A1                 ; Tout changement de ligne est ;
EXPLS1: XCONS   NIL,A1                  ; transformé en espace ;
        MOVE    (@ LEXPLD),A2           ; récupère la liste courante ;
        SCDR    A1,A2                   ; rajoute l'élément en queue ;
        MOVE    A1,(@ LEXPLD)           ; nouvelle queue ;
        POP     A2                    ; restaure les registres de travail ;
        POP     A1,,[RETURN]            ; et rentre ;

            ; Rajoute le caractère A4 dans la liste EXPLODE ;

EXPLCH: PUSH    A4                      ; sauve le caractère ;
        PUSH    A2                      ; pour travailler ;
        XCONS   NIL,A4                  ; transformé en espace ;
        MOVE    (@ LEXPLD),A2           ; récupère la liste courante ;
        SCDR    A4,A2                   ; rajoute l'élément en queue ;
        MOVE    A4,(@ LEXPLD)           ; nouvelle queue ;
        POP     A2                    ; restaure les registres de travail ;
        POP     A4,,[RETURN]            ; et rentre ;
```

```
            ; données des fonctions de sortie ;

BUFAT:   BLOCK  30,0                    ; tampon de sortie d'un atome ;
BUFOUT:  BLOCK  132,'" "                ; tampon de sortie d'une ligne ;

PRMDP:   DATA   100                     ; profondeur maximum d'impression ;
PRCDP:   DATA   0                       ; profondeur courante d'impression ;

PRMLP:   DATA   1000                    ; nb de ligne maximum d'impression ;
PRCLP:   DATA   0                       ; nb courant de lignes imprimées ;

PRMLN:   DATA   1000                    ; longeur maximum d'impression ;
PRCLN:   DATA   0                       ; longeur courante d'impression ;

SPRINT:  DATA   0                       ; SP au début de PROBJ ;
POCOUR:  DATA   0                       ; index courant sur BUFOUT ;
NBLEFT:  DATA   0                       ; longeur de la marge gauche ;
NBRIG:   DATA   70                      ; longeur de la ligne ;

            ; Données de la fonction EXPLODE ;

IEXPLD:  DATA   NIL                     ; indicateur EXPLODE ;
LEXPLD:  DATA   NIL                     ; liste en formation ;


            ))

; fin de l'imprimeur ;

'OK

; Définition du macro-caractère d'activation ;

(DMC "Ɛ" ()
   ; Ɛ = ↑P ;
   [QUOTE
    (VCMC2
      [['MOVE [QUOTE (READ)] 'A1]
       '(NOP NIL NIL CALL (PROBJT))
       '(NOP NIL NIL CALL (PPRINT))
       '(NOP NIL NIL CALL (EXPLODE))])])

; Epilogue standard ;

(PROGN
   (ANACODE '(~pprinter))
   (STATUS 1 1)
   (PRINT "...  Imprimeur VLISP - VCMC2 chargé.")
   (PRINT "     Ɛs pour imprimer s"))
```

CROSS REFERENCE

Signification des codes associés aux numéros des lignes :

- \# définition de fonction de type DE DF DM DMC ou ENTRY
- & définition de fonction de type ESCAPE, ESCLOOP
- : définition d'étiquette dans PROG, DO, LAP ...
- , variable argument d'une fonction
- = variable affectée par SETQ ou SETQQ
- ' nom apparaissant dans une S-expression quotée
- \- code instruction assembleur

⊏	DMC	592
[S	DMC	35
CODE	DF	30
DMP	ENTRY	410
EOL	ENTRY	85
EXPLODE	ENTRY	525
LMARGIN	ENTRY	480
OUTBUF	ENTRY	511
OUTPOS	ENTRY	502
PLENGTH	ENTRY	422
PPRIN	ENTRY	371
PPRINT	ENTRY	372
PRETTY	ENTRY	373
PRIN	ENTRY	204
PRINCH	ENTRY	233
PRINT	ENTRY	216
PRINTLENGTH	ENTRY	456
PRINTLEVEL	ENTRY	446
PRINTLINE	ENTRY	436
PTYPE	ENTRY	428
RMARGIN	ENTRY	490
TERPRI	ENTRY	223

1	*MARK*	474'										
2	0SUBR	85										
3	1SUBR	223	371	372	422	436	446	456	480	490	502	525
4	2SUBR	233	428	511								
5	@	54	55	56	57	58	59	96	103	104	107	107
		108	109	109	110	115	116	116	118	119	124	125
		125	127	128	134	135	152	153	154	155	160	161
		161	176	177	177	193	309	312	327	331	339	350
		355	362	440	441	450	451	460	461	472	484	485
		496	497	506	507	530	531	533	542	544	553	555
6	A1	61	63	64	65	71	75	80	88	99	100	133
		136	138	151	157	159	164	166	170	183	184	184
		185	186	187	206	209	210	211	212	225	226	227
		228	235	250	250	252	255	256	257	257	260	261
		262	263	264	270	272	273	275	289	300	301	311
		325	330	342	343	343	344	345	346	348	349	354
		356	357	358	359	360	361	377	378	382	383	387
		389	390	391	394	396	398	400	403	404	405	412
		413	413	424	424	431	432	432	439	440	441	449
		450	451	459	460	461	470	471	472	476	482	484
		485	492	493	494	495	496	497	504	506	507	516

		517	517	534	538	540	541	543	544	546	596'	
7	A2	72	76	79	98	99	101	102	137	140	141	164
		165	166	167	170	209	212	236	237	238	239	240
		241	258	263	265	274	277	325	326	387	388	389
		391	392	399	400	401	402	404	412	430	431	515
		516	527	528	529	530	539	542	543	545	551	553
		554	556									
8	A3	73	78	138	142	143	167	168	264	267	270	392
		393	395	397	414							
9	A4	74	77	118	133	134	135	140	162	169	172	181
		189	192	235	253	254	267	268	269	271	291	294
		341	347	550	552	554	555	557				
10	ADD	64-	102-	108-	119-	128-	134-	141-	160-	176-	309-	327-
		339-	355-									
11	ANACODE	604										
12	BLOCK	561-	562-									
13	BUFAT	136	139	561:								
14	BUFOUT	60	97	117	126	562:						
15	CALL	76	87	135	140	155	169	172	175	178	180	184
		188	189	190	191	192	206	207	208	218	227	238
		253	258	269	271	273	290	291	292	293	299	300
		310	316	320	326	328	329	335	340	341	344	346
		347	356	358	360	377	382	401	483	505	513	532
		597'	598'	599'								
16	CAR	164-	170-	184-	250-	257-	263-	270-	273-	343-	387-	413-
17	CDR	166-	167-	183-	206-	264-	267-	272-	300-	325-	342-	344-
		356-	358-	412-	534-							
18	CODE	30#	40									
19	CONS	391-	394-	396-	398-	404-	405-					
20	DATA	279-	280-	281-	282-	283-	284-	285-	564-	565-	567-	568-
		570-	571-	573-	574-	575-	576-	580-	581-			
21	DE	394'										
22	DF	30	396'									
23	DM	398'										
24	DMC	35	592									
25	DMP	405'	410#									
26	DP	412:										
27	ENTRY	85	204	216	223	233	371	372	373	410	422	428
		436	446	456	480	490	502	511	525			
28	EOL	75'	85#	87:								
29	EVCAR	206										
30	EVFEXP	261										
31	EXPLCH	115	550:									
32	EXPLODE	525#	527:	599'								
33	EXPLS	96	124	538:								
34	EXPLS1	541:										
35	FALSE	388	390	403								
36	FATOM	388-										
37	FLIST	159-	255-	354-								
38	FNIL	96-	115-	124-	168-	252-	268-					
39	FNUMB	470-	493-									
40	FSUBR	204	216	373	410							
41	FTYP	392-										
42	FULLIN	71:	116	125	135	197	219	328	340	360	383	
43	FVAL	389-										
44	GE	116-	125-	143-								
45	GET	258	401									
46	GT	104-	135-	228-	241-	495-						
47	IEXPLD	54	96	115	124	531	533	580:				
48	ITSOFT	76										
49	JUMP	65	96	98	104	110	115	116	124	125	138	143
		159	161	162	165	168	170	173	177	181	186	187
		197	211	219	225	228	236	237	241	252	254	255

		260	261	265	268	270	289	294	309	311	321	327
		330	339	349	361	383	388	390	393	394	395	396
		397	398	403	405	414	430	439	449	459	470	471
		476	482	492	493	494	495	504	515			
50	JUMPX	277-										
51	L	30,	30									
52	LE	161-	177-	472-								
53	LEXPLD	530	542	544	553	555	581:					
54	LMARG1	482	485:									
55	LMARGIN	480#	482:									
56	LT	65-	471-	494-								
57	MOVE	54-	55-	56-	57-	58-	59-	61-	75-	76-	88-	98-
		107-	137-	153-	154-	155-	162-	169-	172-	181-	189-	192-
		209-	212-	226-	235-	237-	253-	254-	258-	269-	271-	291-
		294-	341-	347-	400-	401-	414-	440-	441-	450-	451-	460-
		461-	484-	485-	496-	497-	506-	507-	527-	530-	531-	533-
		540-	542-	544-	553-	555-	596'					
58	MOVEX	63-	101-	118-	127-	516-						
59	NBLEFT	58	107	309	312	327	331	339	350	355	362	484
		485	575:									
60	NBRIG	59	116	125	135	472	496	497	576:			
61	NEQ	109-	165-	265-	393-	395-	397-					
62	NOP	35'	67-	87-	144-	173-	175-	180-	188-	190-	191-	207-
		208-	218-	219-	229-	242-	290-	292-	293-	299-	310-	316-
		320-	321-	328-	329-	335-	340-	483-	505-	532-	597'	598'
		599'										
63	OK	588'										
64	OUT	100-	105-	106-								
65	OUTBU1	515	517:									
66	OUTBUF	511#	513:	513								
67	OUTCH	115:	140	162	169	172	181	189	192	238	253	254
		269	271	291	294	341	347					
68	OUTLI1	99:	104									
69	OUTLI2	98	104:									
70	OUTLIN	87	96:	227								
71	OUTPO1	504	507:									
72	OUTPOS	502#	504:									
73	OUTSP	124:	175	188	190	207	290	292	299			
74	PLEN	133-	138-	424-								
75	PLENGTH	422#	424:									
76	PNAM	136-										
77	POCOUR	57	103	104	107	116	118	119	125	127	128	134
		506	507	574:								
78	POP	77-	78-	79-	80-	157-	185-	210-	239-	262-	275-	301-
		345-	348-	357-	359-	378-	383-	402-	545-	546-	556-	557-
79	POSER	470	471	474:	493	494	495					
80	PP	251:	270	293	377	382						
81	PP1	299:	310	316	320	326	329	335				
82	PP10	252	255:									
83	PP11	260	262:									
84	PP12	265	268	271:								
85	PPCAR	249:	273	300	344	356	358					
86	PPRIN	371#	377:									
87	PPRINT	372#	382:	394	396	398	405	598'				
88	PPT0	279	309:									
89	PPT02	310:	311									
90	PPT03	309	311:									
91	PPT1	280	316:									
92	PPT2	281	320:									
93	PPT3	282	325:	346								
94	PPT31	321	327:									
95	PPT32	328:	330									
96	PPT33	327	330:									

97	PPT4	283	335:									
98	PPT5	284	339:									
99	PPT51	340:	349									
100	PPT55	339	349:									
101	PPT6	285	354:									
102	PPT61	356:	361									
103	PPTBLX	277	279:									
104	PPTE	276	289:									
105	PPTE7	289	294:									
106	PRAT2	139:	143									
107	PRAT3	138	142:									
108	PRATOM	133:	159	191	255							
109	PRCDP	155	160	161	193	565:						
110	PRCLN	154	176	177	571:							
111	PRCLP	108	109	153	568:							
112	PRETT3	393	395:									
113	PRETT4	395	397:									
114	PRETT6	397	399:									
115	PRETTY	258'	373#	387:	401'	414'						
116	PRII1	63:	65									
117	PRIN	204#	211:	218								
118	PRIN1	35										
119	PRIN11	206:	211									
120	PRINC1	238:	241									
121	PRINC2	236	237	241:								
122	PRINCH	233#	235:									
123	PRININI	54:										
124	PRINT	216#	218:	606	607							
125	PRINTL1	449	451:									
126	PRINTLENGTH	456#	458:									
127	PRINTLEVEL	446#	448:									
128	PRINTLINE	436#	438:									
129	PRINTN1	439	441:	459	461:							
130	PRMDP	55	161	450	451	564:						
131	PRMLN	56	177	460	461	570:						
132	PRMLP	109	440	441	567:							
133	PROBJ	151:	197	208	532							
134	PROBJ0	155	159:	170	184							
135	PROBJ2	161	164:									
136	PROBJ5	165	168	172:								
137	PROBJ6	175:	186									
138	PROBJ7	173	176:									
139	PROBJ8	177	183:									
140	PROBJ9	178	187	192:								
141	PROBJD	178	180	181:								
142	PROBJE	110	156:									
143	PROBJT	197:	597'									
144	PROGN	603										
145	PTYP	274-	432-									
146	PTYP1	430	432:									
147	PTYPE	428#	430:									
148	PUSH	71-	72-	73-	74-	151-	178-	197-	238-	256-	276-	377-
		382-	399-	474-	475-	476-	529-	538-	539-	550-	551-	
149	PUT	414										
150	QUOTE	165'	265'	594	596							
151	READ	35	596									
152	RETURN	67	80	88	109	119	128	144	157	193	212	229
		242	301	312	331	350	354	362	378	424	432	441
		451	461	472	485	497	507	517	534	546	557	
153	REVIVE	66-										
154	RMARG1	492	497:									
155	RMARGIN	490#	492:									
156	RPLACA	39										

157	SCDR	543-	554-									
158	SERROR	476										
159	SILENCE	62-										
160	SINDEX	60-	97-	117-	126-	139-	513-					
161	SPRINT	110	152	573:								
162	SPTYP	431-										
163	SSTACK	110-										
164	STACK	152-										
165	STATUS	26	605									
166	SUB	103-	142-	193-	227-	240-	312-	331-	350-	362-		
167	T	531'										
168	TERPR1	227:	228									
169	TERPR2	225	228:									
170	TERPRI	223#	225:									
171	TESPOS	470:	483	505	513							
172	TLIST	186-	211-	311-	330-	346-	349-	360-	361-			
173	TNIL	187-	260-	289-	326-	390-	403-	430-	439-	449-	459-	482-
		492-	504-	515-								
174	TNUMB	225-	236-									
175	TST	183	206	272	273	274	300	342	344	356	358	534
176	VCMC2	595										
177	XCONS	528-	541-	552-								
178	XMOVE	99-	140-	517-								
179	XTOPST	261-										
180	~pprinter	39'	604'									

APPENDICE F

```
;                  V C M C 2 I   .   V L I                        ;
;                                                                 ;
;          INTERPRETE VLISP en machine VCMC2                      ;
;-----------------------------------------------------------------;
;        Ce fichier doit être édité au moyen de :                 ;
;            (CROSSF V2I T T T {NIL/T} T T)                       ;
;              {NIL/T} si VERSATEC                                ;
;-----------------------------------------------------------------;
;                                                                 ;
;                  Jérôme CHAILLOUX                               ;
;                                                                 ;
;        Département d'Informatique                               ;
;        Université de Paris 8 - Vincennes                        ;
;        Route de la Tourelle 75571 Paris Cédex 12                ;
;        Tél : 374 12 50 poste 299                                ;
;                                                                 ;
;        L.I.T.P.   (CNRS LA 248)                                 ;
;        2 Place Jussieu 75221 Paris Cédex 05                     ;
;        Tél : 336 25 25 poste 53-70                              ;
;                                                                 ;
;        I.R.C.A.M.                                               ;
;        31 Rue St Merri 75004 Paris                              ;
;        Tél : 277 12 33 poste 48-48                              ;
;-----------------------------------------------------------------;
;                                                                 ;
; règles de reconnaissance des identificateurs :                  ;
;                                                                 ;
;   1er car.          signification                               ;
;                                                                 ;
;      &        fonctions d'échappements (ESCAPEs)                ;
;      *        fonctions internes du simulateur                  ;
;      ~        variables globales à tout le simulateur           ;
;      #        variables libres pour certaines fonctions         ;
;               (mais liees par des fnts de l'interprète)         ;
;      !        indicateurs sur P-listes                          ;
;      ?        indicateurs du simulateur  (e.g. T ou NIL)        ;
;                                                                 ;
;-----------------------------------------------------------------;

(STATUS 2 1 2)

; Pour prettyprinter correctement le code ;

(DF CODE (L) L)
```

```
; Pour contrôler la lecture ;

(DMC "S" () ; (PRIN1 (READ)) ; '(NOP))
```

```
; TOP-LEVEL de l'interprète et erreurs ;

(RPLACA '~interpreter
 (CODE NIL
         ;┌─────────────────────────────────────────────┐;
         ;│                                             │;
         ;│          Interprète VLISP - VCMC2           │;
         ;│                                             │;
         ;└─────────────────────────────────────────────┘;

         ; Erreur de la machine ;

MERROR:  PRINI   '"***** Erreur machine : HALT VCMC2."
         NOP     ,,[CALL (OUTLIN)]
         PRSTACK'100
         PRSTAT
         STOP

         ; Erreur de l'interprète ;

SERROR:  MOVE    'ERROR,A1                  ; nom de la fonction ;
         MOVE    NIL,A2                     ; init liste des arguments ;
SERR01:  POP     A3                         ; argument suivant ;
         EQ      A3,'*MARK*,[JUMP (SERR02)] ; c'est fini ;
         CONS    A3,A2,[JUMP (SERR01)]  ; pour les autres arguments ;
SERR02:  CONS    (@ FORME),A2,[JUMP (ITSOFT)] ; rajoute FORME en tête ;

         ; la fonction ERROR standard ;

         ENTRY   ERROR,NSUBR

ERROR:                                      ; erreur normale ;
         CAR     A1,TST                     ; empile la dernière forme ;
         CDR     A1,TST,[CALL (OUTLIN)]; empile le reste des args ;
         PRINI   '"***  ",,[JUMP (ERR04)]
ERR02:   CDR     A1,TST                     ; sauve le reste ;
         CAR     A1,A1,[CALL (PROBJ)]       ; édite l'objet suivant ;
ERR04:   POP     A1
         TLIST   A1,,[JUMP (ERR02)]         ; il en reste ;
         NOP     ,,[CALL (OUTLIN)]
         PRINI   '"Dernière forme évaluée : "
         POP     A1,,[CALL (PROBJT)]        ; imprime la dernière forme ;
         PRSTACK'30
         MOVE    NIL,A1,[JUMP (EXITCHRONOLOG)]
```

```
                ; TOPLEVEL ;
                ;-----------;

                ; appel EVAL 1 fois ;

        SINGLE:                                         ; EVAL 1 fois ;
        SINGLET:                                        ; peut-être avec TRACE ;
                SSTACK '(((STOP)) *EOS*)
                SILENCE
                MOVE    NIL,(@ EVALST)                  ; plus de step ;
                MOVE    '0,(@ EVALCH)                   ; chrono 0 ;
                PUSH    A1,,[CALL (READINI)]            ; sauve l'argument à évaluer ;
                NOP     ,,[CALL (PRININI)]
                MOVE    '0,(@ PBIND)
                MOVE    NIL,(@ FORME)
                PRINI   '"? "
                TOPST   A1,,[CALL (PROBJT)]             ; imprime l'argument ;
                REVIVE
                POP     A1,,[CALL (EVALA1)]             ; évalue l'argument ;
                SCVAL   A1,'IT                          ; le IT feature ;
                SILENCE
                PRINI   '"= ",,[CALL (PROBJT)]
                REVIVE
                NOP     ,,[RETURN]

                ; Boucle principale de l'interprète ;

        MAIN:   SSTACK '(((STOP)) *EOS*)
                MOVE    NIL,(@ EVALST)                  ; plus de step ;
                NOP     ,,[CALL (READINI)]
                NOP     ,,[CALL (PRININI)]
                MOVE    '0,(@ PBIND)
                MOVE    NIL,(@ FORME)
        MAIN1:  MOVE    '0,(@ EVALCH)                   ; CHRONOLOGIE = 0 ;
                MOVE    'TOPLEVEL,A1                    ; le nom de la fonction ;
                MOVE    NIL,A2,[CALL (ITSOFT)]          ; la liste d'arguments ;
                PRSTAT                                  ; impression des statistiques ;
                NOP     ,,[JUMP (MAIN1)]                ; retourne à l'interprète ;

                ; fonction TOP-LEVEL ;

                ENTRY   TOPLEVEL,0SUBR

        TOPLEVEL:
                SILENCE                                 ; pour éviter les lères traces ;
                NOP     ,,[CALL (OUTLIN)]
                PRINI   '"toplevel",,[CALL (OUTLIN)]
                PRINI   '"?"
                REVIVE
                NOP     ,,[CALL (READ)]
                NOP     ,,[CALL (PROBJT)]               ;    (PRINT 'TOPLEVEL)       ;
                NOP     ,,[CALL (EVALA1)]               ;    (PRINT (EVAL (READ)))) ;
                SCVAL   A1,'IT                          ; le IT feature ;
                SILENCE
                PRINI   '"= ",,[CALL (PROBJT)]
                REVIVE
                NOP     ,,[RETURN]
```

```
            ; Interprète : EVAL ;
            ;——————————————————;
            ; EVAL : 3SUBR   ;
            ; A1 ← la forme à évaluer ;
            ; A2 ← la valeur de CHRONOLOGIE ;
            ; A3 ← la valeur de STEPEVAL ;
            ENTRY   EVAL,3SUBR
EVAL:       TNIL    A2,,[JUMP (EVALL2)]       ; pas de nouvel état ;
            PUSH    (@ EVALST)                ; sauve le step ;
            PUSH    (@ EVALCH)                ; sauve l'état ;
            MOVE    A2,(@ EVALCH),[JUMP (EVALL3)] ; pour finir le bloc ;
EVALL2:     TNIL    A3,,[JUMP (EVALAN)]       ; pas d'arguments spéciaux ;
            PUSH    (@ EVALST)                ; sauve l'état de la trace ;
            PUSH    (@ EVALCH)                ; sauve l'état de CHRONOLOGIE ;
EVALL3:     MOVE    A3,(@ EVALST)             ; charge la TRACE ;
            PUSH    (@ PBIND)                 ; sauve l'ancien PBIND ;
            PUSH    '3                        ; type du bloc ;
            STACK   (@ PBIND)                 ; nouveau PBIND ;
            PUSH    (UNBIND),,[JUMP (EVALAN)]
            ; EVAL interne ;
EVAL1:                                     ;*** (EVAL (CAR A1)) pour les 1SUBR ;
EVCAR:                                        ;*** (EVAL (CAR A1)) interne ;
            CAR     A1,A1                     ; A1 ← (CAR A1) ;
EVALA1:                                     ;*** (EVAL A1) avec test interruption ;
            FNIL    (@ EVALST),,[JUMP (EVALT)] ; il y a la TRACE ;
EVALAN:                                     ; *** (EVAL A1) sans test interruption ;
            MOVE    A1,(@ FORME)              ; sauve toute la forme ;
            DISPT   (TEVAL1),A1               ; dispatch sur type ;
TEVAL1:     DATA    (EVALAT)                  ; la forme est un symbole ;
            DATA    (EVALNB)                  ; la forme est un nombre ;
            DATA    (EVALIS)                  ; la forme est une liste ;
            ; Evaluation des symboles atomiques ;
EVALAT:     CVAL    A1,A1                     ; charge la valeur de l'atome ;
            NEQ     A1,'~UNDEF,[RETURN]       ; test si la C-VAL est définie ;
            PUSH    '*MARK*                   ; bloc de fin des arguments ;
            PUSH    '"EVAL : Variable indéfinie.",,[JUMP (SERROR)]
            ; Evaluation des nombres ;
EVALNB:     NOP     ,,[RETURN]                ; les nombres sont des constantes ;
```

```
        ; Les interruptions de EVAL : EVALT STEPEVAL ITSOFT ;
        ;-----------------------------------------------------;

EVALT:                                  ; si TRACEVAL non-NIL ;
        MOVE    NIL,(@ EVALST)          ; efface la trace ;
        MOVE    A1,A2                   ; prépare la liste d'arguments;
        XCONS   NIL,A2                  ; A2 ← (forme) ;
        MOVE    'STEPEVAL,A1,[CALL (ITSOFT)]
                                        ; appel de (STEPEVAL forme) ;
        MOVE    'T,(@ EVALST),[RETURN]

        ; La fonctions STEPEVAL standard ;
        ; réalise une trace de tous les appels internes de EVAL ;
        ; l'argument est la forme quidevait être évaluée ;

        ENTRY   STEPEVAL,1SUBR

STEPEVAL:
        PUSH    A1                      ; sauve la forme ;
        MOVE    '"->",A1,[CALL (PROBJ)] ; imprime -> ;
        TOPST   A1,,[CALL (PROBJT)]     ; puis l'argument ;
        POP     A1                      ; récupère la forme ;
        MOVE    (@ EVALCH),A2           ; effectue ;
        SUB     '1,A2                   ; (SUB1 (STATUSEVAL)) ;
        MOVE    'T,A3,[CALL (EVAL)]     ; trace l'évaluation de la forme ;
        PUSH    A1                      ; sauve la valeur de l'évaluation ;
        MOVE    '"<-",A1,[CALL (PROBJ)] ; imprime <-;
        TOPST   A1,,[CALL (PROBJT)]     ; imprime la valeur ;
        POP     A1,,[RETURN]            ; et rentre de STEPEVAL ;

        ; Lancement d'une IT soft ;
        ; A1 ← la fonction ;
        ; A2 ← la liste d'arguments (a la APPLY) ;
        ; change de CHRONOLOGIE  +1  ;

ITSOFT:                                 ; prépare un bloc 3 ;
        PUSH    (@ EVALST)              ; sauve l'état de la trace courante ;
        PUSH    (@ EVALCH)              ; sauve la chronologie courante ;
        PUSH    (@ PBIND)               ; sauve l'ancien PBIND ;
        PUSH    '3                      ; type de bloc = 3 ;
        STACK   (@ PBIND)               ; new PBIND ;
        ADD     '1,(@ EVALCH)           ; change de CHRONOLGIE ;
        PUSH    (UNBIND),,[JUMP (APPLY)] ; et appel APPLY ;
```

```
        ; Evaluation des formes ;
        ;-----------------------;

EVALIS:                                 ;*** Evalue la forme A1 ;
        CAR     A1,A2                   ; A2 ← la fonction, ;
        CDR     A1,A1                   ; A1 ← la liste des arguments ;
EVALFU:                                 ;*** Evalue la fnt A2 et larg A1 ;
        DISPT   (TEVAL2),A2             ; dispatch sur la fonction ;

TEVAL2: DATA    (EVALFAT)               ; la fonction est un symbole ;
        DATA    (EVALFNB)               ; la fonction est un nombre ;
        DATA    (EVALFLI)               ; la fonction est une liste ;

EVALFAT:FVAL    A2,TST            ; empile la F-VAL de l'atome fonction ;
        FTYP    A2,A3           ; A3 ← le F-TYP codé de l'atome fonction ;
EVALIN:                                 ;*** Evalue un INTERNAL ;
        JUMPX   (TEVAL3),A3             ; branchement indirect indexé ;

        ; Table du branchement indirect indexé  sur le F-TYP de EVAL ;
        ; On effectue une rupture de séquence avec ;
        ; - A1 ← la liste des arguments ;
        ; - A2 ← le nom de la fonction ;
        ; - A3 ← le F-TYPE codé de la fonction ;
        ; - pile ← la F-VAL de la fonction ;

TEVAL3: DATA    (UDFE)          ; code 0 : pas de définition de fonction ;
        DATA    (EVAL0)                 ; code 1 : SUBR à 0 argument ;
        DATA    (EVAL1)                 ; code 2 : SUBR à 1 argument ;
        DATA    (EVAL2)                 ; code 3 : SUBR à 2 arguments ;
        DATA    (EVAL3)                 ; code 4 : SUBR à 3 arguments ;
        DATA    (EVALN)                 ; code 5 : SUBR à N argumemnts ;
        DATA    (EVALF)                 ; code 6 : FSUBR ;
        DATA    (EVEXP)                 ; code 7 : EXPR ;
        DATA    (EVFEXP)                ; code 8 : FEXPR ;
        DATA    (EVMAC)                 ; code 9 : MACRO ;
        DATA    (EVESC)                 ; code 10 : ESCAPE ;

        ; La fonction est un nombre == CNTH ;

EVALFNB:PUSH    A2,,[CALL (EVCAR)]      ; sauve le nb et évalue la liste ;
        POP     A2                      ; A2 ← le nombre ;
        LE      A2,'0,[JUMP (CAR)]      ; C'est fini : 1er élément ;
        NOP     ,,[JUMP (EVALN2)]       ; recherche de l'élément ;
EVALN1: CDR     A1,A1                   ; avance dans la liste ;
        FLIST   A1,,[RETURN]            ; si la liste est vide ;
EVALN2: SUBFZ   '1,A2,[JUMP (EVALN1)]   ; il faut encore avancer ;
        CAR     A1,A1,[RETURN]          ; ramène l'élément pointé ;

        ; Evaluation des fonctions spéciales ;

EVALFLI:                                ;*** la fonction est spéciale ;
        CAR     A2,A3                   ; A3 ← fonction de la fonction ;
        EQ      A3,'LAMBDA,[JUMP (EVALL)] ; c'est une lambda-explicite ;
        EQ      A3,'INTERNAL,[JUMP (EVALI)]
                                        ; c'est une fonction INTERNAL ;
        PUSH    A1                ; sinon sauve la liste des arguments ;
        MOVE    A2,A1,[CALL (EVALA1)]   ; évalue la fonction ;
        MOVE    A1,A2                   ; A2 ← la nouvelle fonction ;
        POP     A1,,[JUMP (EVALFU)]  ; vers la re-évaluation de la forme ;

EVALL:                                ;*** la fonction est une λ explicite ;
        CDR     A2,TST,[JUMP (EVEXP)]
```

```

EVALI:                                      ;*** la fonction est INTERNAL ;
        CDR     A2,A2                       ; A2 ← (FTYP FVAL) ;
        CAR     A2,A3                       ; A3 ← FTYP ;
        CDR     A2,A2                       ; A2 ← (FVAL) ;
        CAR     A2,TST,[JUMP (EVALIN)]      ; pile ← FVAL ;

```

```
                ; Evaluation rapide de type SUBR ;
                ;-------------------------------;

UDFE:
                ; l'atome n'a pas de fonction associée ;
                ; indirection sur la C-VAL ;

        POP     A3                      ; nettoie l'ancienne F-VAL ;
        CVAL    A2,A3                   ; indirection sur C-VAL ;
        CVAL    A3,A4                 ; teste si cette C-VAL n'est pas une ;
        EQ      A3,A4,[JUMP (UDFER)]    ; constante : evite le bouclage ;
        MOVE    A3,A2,[JUMP (EVALFU)]   ; et reappelle la fonction ;
UDFER:  PUSH    '*MARK*                 ; fin des arguments ;
        PUSH    '"EVAL : fonction indéfinie : "
        PUSH    A2,,[JUMP (SERROR)]

POPJ:   NOP     ,,[RETURN]              ; continuation RETURN simple ;

EVAL0:  NOP     ,,[RETURN]              ;** Traitement des 0SUBR ;
                                        ; on tombe sur la F-VAL empilée! ;

EVAL2:                                  ;*** Traitement des 2SUBR ;
        CDR     A1,TST,[CALL (EVCAR)]   ; évalue le 1er argument ;
        XTOPST  A1,,[CALL (EVCAR)]      ; évalue le 2ème argument ;
        MOVE    A1,A2                   ; A2 ← la valeur du 2ème arg ;
        POP     A1,,[RETURN]            ; A1 ← la valeur du 1er arg ;
                                        ; on tombe sur la F-VAL empilée! ;

EVAL3:                                  ;*** Traitement des 3SUBR ;
        CDR     A1,TST,[CALL (EVCAR)]   ; évalue le 1er argument ;
        XTOPST  A1                      ; echange sa valeur et le reste ;
        CDR     A1,TST,[CALL (EVCAR)]   ; évalue le 2ème argument ;
        XTOPST  A1,,[CALL (EVCAR)]      ; évalue le 3ème argument ;
        MOVE    A1,A3                   ; A3 ← la valeur du 3ème argument ;
        POP     A2                      ; A2 ← la valeur du 2ème argument ;
        POP     A1,,[RETURN]            ; A1 ← la valeur du 1er argument ;

EVALF:  NOP     ,,[RETURN]              ;** Traitement des FSUBR ;
                                        ; on tombe sur la F-VAL empilée! ;
```

```
            ; Evaluation des EXPR ;
            ;--------------------;

            ; Fabrique dans la pile un bloc de contrôle de type ;

            ;                         [         UNBIND          ] ;
            ;     PBIND  →            [           0             ] ;
            ;                         [ F-VAL de la fonction    ] ;
            ;                         [ fin du bloc de contrôle ] ;
            ;                         [ nom de la variable N    ] ;
            ;                         [ son ancienne valeur     ] ;
            ;                              ..........             ;
            ;                              ..........             ;
            ;                         [ nom de la variable 1    ] ;
            ;                         [ son ancienne valeur     ] ;
            ;                         [        ' *MARK*         ] ;
            ;                         [      ancien PBIND       ] ;
            ;                          -------------------------- ;

EVEXP:  TOPST  A4                      ; A4 ← la F-VAL ;
        CAR    A4,A2                   ; A2 ← la liste des paramètres ;
        PUSH   '*MARK*,, [JUMP (EVEXP2)]
                               ; empile le marqueur de fin de variables ;

        ;  Fabrication des emplacements en pile et évaluation ;

EVEXP1: CDR    A1,TST                  ; empile le reste des valeurs ;
        PUSH   A2,, [CALL (EVCAR)]     ; empile toutes les variables ;
        POP    A2                      ; récupère toutes les variables ;
        XTOPST A1                      ; force la valeur évaluée ;
        CAR    A2,TST                  ; empile le nom de la variable ;
        CDR    A2,A2                   ; variable suivante  ;
EVEXP2: TLIST  A2,, [JUMP (EVEXP1)]    ; il en reste ;
        TNIL   A2,, [JUMP (EVEXP3)]    ; c'est vraiment la fin ;
        PUSH   A2,, [CALL (EVLIS)]     ; c'est une variable pointée ;
        XTOPST A1                      ; empile la valeur ;
        PUSH   A1                      ; empile le nom ;

        ;  Effectue la liaison superficielle ;

EVEXP3: STACK  A4,, [JUMP (EVEXP5)]    ; garde la hauteur de la pile ;
EVEXP4: CVAL   A2,A1                   ; récupère l'ancienne C-VAL ;
        XTOPST A1                      ; qui est sauvée en pile ;
        SCVAL  A1,A2                   ; effectue la nouvelle liaison ;
        POP    A1                      ; saute la valeur ;
EVEXP5: POP    A2                     ; lecture de la nouvelle variable ;
        NEQ    A2,'*MARK*, [JUMP (EVEXP4)] ; il y en a encore ;
        MOVE   (@ PBIND),A1            ; A1 ← le PBIND ;
        XTOPST A1                  ; force PBIND  et récupère la F-VAL ;

        ; Test de récursion terminale ou mutuelle ;

        POP    A2                      ; saute l'ancien PBIND ;
        STACK  A3                      ; A3 ← pointe sur la fin du bloc ;
EVEXP6: NEQ    TST, (UNBIND), [JUMP (EVEXPN)]
                                        ;  pas une évaluation terminale ;
        JUMPX  (TEVEX),TST             ; aiguillage sur le type de bloc ;

TEVEX:  DATA   (TEVEXL)                ; type 0 : bloc LAMBDA ;
        DATA   (TEVEXW)                ; type 1 : bloc WHERE/ESCAPE ;
        DATA   (EVEXPN)                ; type 2 : bloc LETF ;
        DATA   (EVEXPN)                ; type 3 : bloc STEPEVAL ;
```

```
TEVEXL:                                        ; bloc LAMBDA ;
          EQ      A1,TST,[JUMP (EVEXP9)]
                                               ; c'est récursif : prépare la pile ;
          SSTACK  TST,,[JUMP (EVEXP6)]         ; passe à la fin du bloc ;

TEVEXW:                                        ; bloc WHERE/ESCAPE ;
          POP     A2                           ; saute un élément ;
          POP     A2                           ; saute un élément ;
          POP     A2                           ; saute un élément ;
          POP     A2,,[JUMP (EVEXP6)]          ; saute l'ancien PBIND ;

EVEXP9:                                        ; c'est tail ou co-post récursif ;
          SSTACK  A3                           ; remise état initial de la pile ;
          CDR     A1,A1,[JUMP (PROGN)]         ; et réalise un JUMP! ;

EVEXPN:                                        ; appel normal ;
          SSTACK  A4                           ; repasse en début de bloc ;
          PUSH    A3                           ; empile l'adresse de fin du bloc ;
EVEXPF:   PUSH    A1                           ; empile la F-VAL ;
          PUSH    '0                           ; empile le code du bloc ;
          STACK   (@ PBIND)                    ; sauve le pointeur de pile ;

          ; exécution du corps de la fonction ;

EVEXPG:   CDR     A1,A1,[CALL (PROGN)]         ; évalue le corps ;
                                               ;... UNBIND doit suivre ...;
```

```
                ; UNBIND : Délie un bloc de contrôle ;
                ; A3 ← adresse de retour ;
                ; ne doit pas détruire A1 et A2 ;

UNBIND:                                 ;*** s'il faut faire un RETURN final ;
        MOVE    (POPJ),A3               ; prépare un RETURN final ;
UNBINP:                                 ;*** si A3 est prêt ;
        JUMPX   (TUNBD1),TST            ; en fonction du type du bloc ;

TUNBD1: DATA    (UNBDL)                 ; type bloc 0 : bloc lambda ;
        DATA    (UNBDW)                 ; type bloc 1 : bloc where/escape ;
        DATA    (UNBDF)                 ; type bloc 2 : bloc LETF ;
        DATA    (UNBDS)                 ; type bloc 3 : bloc STEPEVAL ;

                ; délie un bloc lambda ;

UNBDL:  POP     A4                      ; enlève l'ancienne F-VAL ;
        POP     A4                      ; enlève la fin du bloc ;
        POP     A4,,[JUMP (UNBDL2)]     ; pour l'analyser ;
UNBDL1: SCVAL   TST,A4                  ; charge l'ancienne C-VAL ;
        POP     A4                      ; dépile encore ;
UNBDL2: NEQ     A4,'*MARK*,[JUMP (UNBDL1)] ; ce n'est pas le marqueur ;
        POP     (@ PBIND)               ; repositionne PBIND ;
        PUSH    A3,,[RETURN]            ; rentre dans A3 ;

                ; délie un bloc  WHERE/ESCAPE ;
                ; Retourne dans A4 le nom de la fonction ;

UNBDW:  POP     (@ PBIND)               ; repositionne PBIND ;
        POP     A4                      ; enlève le nom de la fonction ;
        SFVAL   TST,A4                  ; restaure l'ancienne F-VAL ;
        SFTYP   TST,A4                  ; restaure l'ancien F-TYP ;
        PUSH    A3,,[RETURN]            ; rentre dans A3 ;

                ; délie un bloc LETF ;

UNBDF:  POP     (@ PBIND)               ; restaure l'ancien PBIND ;
        POP     A4                      ; A4 ← le nom ;
        XTOPST  A3                      ; adre ret ↔ (val) ;
        PUSH    A2                      ; sauve tout (APPLY) ;
        PUSH    A1                      ; sauve la valeur ;
        MOVE    A3,A2                   ; A2 ←  la (val) ;
        MOVE    A4,A1,[CALL (APPLY)]    ; Avanti ;
        POP     A1                      ; retourne la valeur du corps ;
        POP     A2,,[RETURN]            ; libere A2 et rentre ;

                ; délie un bloc STEPEVAL ;

UNBDS:  POP     (@ PBIND)               ; restaure l'ancien PBIND ;
        POP     (@ EVALCH)              ; restaure la CHRONOLOGIE ;
        POP     (@ EVALST)              ; restaure la vielle val de STEP ;
        PUSH    A3,,[RETURN]            ; rentre dans A3 ;
```

```
                ; Evaluation des FEXPR, MACRO et ESCAPE ;
                ;--------------------------------------;

EVMAC:   MOVE   (@ FORME),A1          ; l'argument est la forme elle-même ;
         POP    A4                     ; recupère la F-VAL ;
         PUSH   (EVALA1),,[JUMP (EVFEXB)] ; prepare la re-evaluation ;

EVFEXP:  POP    A4                     ; A4 ← la F-VAL empilée ;
EVFEXB:                                ; liaison de type FSUBR/MACRO ;
                ; lie le 1er argument de la FVAL A4 avec  la valeur A1 ;
         CAR    A4,A2                  ; A2 ← la liste de variables ;
         STACK  (@ SAVEP)           ; sauve temporairement la fin du bloc ;
         PUSH   (@ PBIND)              ; prépare le bloc de contrôle ;
         PUSH   '*MARK*,,[JUMP (EVFEX3)]
EVFEX2:  CAR    A2,A3                  ; A3 ← variable suivante ;
         CVAL   A3,TST                 ; sauve l'ancienne CVAL ;
         SCVAL  A1,A3                  ; force la nouvelle valeur ;
         PUSH   A3                     ; sauve le nom de la variable ;
         MOVE   NIL,A1               ; toutes les autres valeurs à NIL ;
         CDR    A2,A2                  ;;
EVFEX3:  TLIST  A2,,[JUMP (EVFEX2)]    ; il reste des variables ;
         PUSH   (@ SAVEP)            ; sauve la fin du bloc de contrôle ;
         MOVE   A4,A1,[JUMP (EVEXPF)]  ; A1 ← la FVAL (cf: EVEXP) ;

         ; Evaluation des ESCAPE ;

EVESC:                                 ;*** de l'appel rapide ;
         PUSH   A2,,[CALL (PROGN)]     ; sauve le nom du ESCAPE ;
         POP    A2                     ; recupère le nom ;
EVESC1:                                ; délie un block dans la pile ;
         EQ     '0,(@ PBIND),[JUMP (ERESC)] ; il n'y a plus de bloc !?! ;
         SSTACK (@ PBIND)              ; pile en début de bloc ;
         TOPST  A3                     ; récupère le sommet de pile ;
         EQ     A3,'1,[JUMP (EVESC3)]  ; c'est un bloc WHERE/ESCAPE ;
         MOVE   (EVESC1),A3,[JUMP (UNBINP)] ; delie ce bloc ;
EVESC3:  MOVE   (EVESC4),A3,[JUMP (UNBINP)]
                                           ; délie ce bloc WHERE/ESCAPE ;
EVESC4:  NEQ    A2,A4,[JUMP (EVESC1)]  ; c'est pas la bonne fonction ;
         NOP    ,,[RETURN]             ; c'est tout bon ;


ERESC:                                 ;*** Erreur dans un ESCAPE ;
         PUSH   '*MARK*                ; fin des arguments ;
         PUSH   '"Erreur ESCAPE : "    ; le message ;
         PUSH   A2,,[JUMP (SERROR)]    ; le nom du ESCAPE ;
```

```
             ; LOSING APPLY ;
             ;-------------;
                                                   ; (APPLY fn larg)   2SUBR ;
                                                   ; (APPLYN fn a1 ... aN)   NSUBR ;
             ENTRY   APPLY,2SUBR
             ENTRY   APPLYN,NSUBR,2
    APPLYN:  CDR     A1,A2                         ; A2 ← la liste des arguments ;
             CAR     A1,A1                         ; A1 ← la fonction ;
    APPLY:   MOVE    A1,A4                         ; en cas d'erreur ;
             FATOM   A1,,[JUMP (APPLYS)]           ; la fonction est spéciale ;
             FVAL    A1,TST                        ; empile la FVAL ;
             FTYP    A1,A3                         ; A3 ← le FTYP codé ;
             MOVE    A2,A1                         ; A1 ← la liste des arguments ;
    APPLIN:                                        ; APPLY INTERNAL ;
             JUMPX   (TAPPLY),A3                   ; branchement indirect indexé ;
             ; Table des branchements indirects indexés ;
             ; pour le LOSING APPLY ;
    TAPPLY:  DATA    (UDFA)                        ; code 0 : pas de définition ;
             DATA    (POPJ)                        ; code 1 : SUBR à 0 argument ;
             DATA    (APPLY1)                      ; code 2 : SUBR à 1 argument ;
             DATA    (APPLY2)                      ; code 3 : SUBR à 2 arguments ;
             DATA    (APPLY3)                      ; code 4 : SUBR à 3 arguments ;
             DATA    (POPJ)                        ; code 5 : SUBR à N arguments ;
             DATA    (POPJ)                        ; code 6 : FSUBR ;
             DATA    (APPEXP)                      ; code 7 : EXPR ;
             DATA    (EVFEXP)                      ; code 8 : FEXPR ;
             DATA    (EVMAC)                       ; code 9 : MACRO ;
             DATA    (EVESC)                       ; code 10 : ESCAPE ;
    UDFA:    POP     A3                            ; libere la F-val ;
    UDFA1:   PUSH    '*MARK*                       ; fin des arguments ;
             PUSH    '"APPLY : fonction indéfinie : " ; le message ;
             PUSH    A4,,[JUMP (SERROR)]           ; empile le nom ;
             ; Evaluation des fonctions spéciales ;
    APPLYS:  TNUMB   A1,,[JUMP (UDFA1)]            ; si c'est un nombre ;
             CAR     A1,A3                         ; A2 ← fonction de la fonction ;
             EQ      A3,'LAMBDA,[JUMP (APPLYL)]    ; c'est une lambda-explicite ;
             EQ      A3,'INTERNAL,[JUMP (APPLYI)]  ; c'est une INTERNAL ;
             PUSH    A2,,[CALL (EVALA1)]
                                                 ; sinon sauve la liste des arguments ;
             POP     A1,,[JUMP (APPLY)]            ; et reappelle APPLY ;
    APPLYL:                                        ; C'est une λ explicite ;
             CDR     A1,TST                        ; doit empiler la FVAL ;
             MOVE    A2,A1,[JUMP (APPEXP)]         ; pour traiter comme une EXPR ;
    APPLYI:                                        ; C'est une INTERNAL ;
             CDR     A1,A1                         ; A1 ← (FTYP FVAL) ;
             CAR     A1,A3                         ; A3 ← FTYP ;
             CDR     A1,A1                         ; A1 ← (FVAL) ;
             CAR     A1,TST,[JUMP (APPLIN)]        ; vers l'INTERNAL ;
             ; Evaluation rapide de type SUBR ;
```

```
APPLY1: CAR     A1,A1,[RETURN]

APPLY2: CDR     A1,A2
        CAR     A1,A1
        CAR     A2,A2,[RETURN]

APPLY3: CDR     A1,A2
        CAR     A1,A1
        CDR     A2,A3
        CAR     A2,A2
        CAR     A3,A3,[RETURN]

        ; Application des EXPR ;
        ; A1 ← liste des valeurs ;
        ; FVAL empilée ;

APPEXP: POP     A2                      ; A2 ← la FVAL ;
        STACK   (@ SAVEP)               ; sauve le pointeur courant ;
        PUSH    (@ PBIND)               ; fabrique le bloc de contrôle ;
        PUSH    '*MARK*
        CAR     A2,A3,[JUMP (APPEX2)]   ; A3 ← liste des variables ;
APPEX1: CAR     A3,A4                   ; A4 ← la variable suivante ;
        CVAL    A4,TST                  ; empile l'ancienne valeur ;
        PUSH    A3                      ; sauve toutes les variables ;
        CAR     A1,A3                   ; A3 ← la nouvelle valeur ;
        CDR     A1,A1                   ; avance dans les valeurs ;
        SCVAL   A3,A4                   ; force la nouvelle valeur ;
        XTOPST  A4                     ; recupère la liste des variables ;
        CDR     A4,A3                   ; avance dans cette liste ;
APPEX2: TLIST   A3,,[JUMP (APPEX1)]     ; il en reste ;
        TNIL    A3,,[JUMP (APPEX3)]     ; c'est la vraie fin ;
        CVAL    A3,TST                  ; empile l'ancienne valeur ;
        PUSH    A3                      ; empile le nom ;
        SCVAL   A1,A3                   ; force la nouvelle CVAL ;
APPEX3: PUSH    (@ SAVEP)           ; empile la fin du bloc de contrôle ;
        MOVE    A2,A1,[JUMP (EVEXPF)]   ; A1 ← la FVAL (cf: EVEXP) ;
```

```
            ; Autres fonctions interprète ;
            ;------------------------------;

            ; (QUOTE s)   FSUBR ;
            ; identique à CAR ;

            ENTRY   QUOTE,FSUBR

QUOTE:      CAR     A1,A1,[RETURN]

            ; (LAMBDA svar e1 ... eN)   FSUBR ;
            ; (INTERNAL f-type f-val)   FSUBR ;
            ; ramènent la forme elle-même ;

            ENTRY   LAMBDA,FSUBR,2
            ENTRY   INTERNAL,FSUBR

LAMBDA:
INTERNAL:
            MOVE    (@ FORME),A1,[RETURN]

            ; (PROGN e1 ... eN)   FSUBR ;
            ; (EPROGN 1)   1SUBR ;

            ENTRY   PROGN,FSUBR,3
            ENTRY   EPROGN,1SUBR
            ; Permet de traiter les fonctions tail-récursives ;

PROGNA3:MOVE    A3,A1                   ; fonction (PROGN A3) interne ;
EPROGN:
PROGN:      CDR     A1,A2                   ; A2 ← reste des éléments ;
            FLIST   A2,,[JUMP (EVCAR)]      ; c'est le dernier élément ;
            PUSH    A2,,[CALL (EVCAR)]   ; sauve le reste et évaue l'élément ;
            POP     A1,,[JUMP (PROGN)]      ; recupère le reste et roule ;

            ; (PROG1 e1 ... eN)   FSUBR ;
            ; évalue une liste et retourne la 1ère valeur ;

            ENTRY   PROG1,FSUBR,3

PROG1:      CDR     A1,TST,[CALL (EVCAR)]  ; évalue le 1er élément ;
            XTOPST  A1,,[CALL (PROGN)]      ; évalue le reste ;
            POP     A1,,[RETURN]            ; retourne la 1ère valeur ;

            ; (LIST e1 ... eN)   FSUBR ;
            ; (EVLIS 1)   1SUBR ;
            ; fabrique une liste de valeurs de manière itérative ;

            ENTRY   LIST,FSUBR
            ENTRY   EVLIS,1SUBR

EVALN:                                      ; évalue les NSUBR ;
LIST:
EVLIS:      FLIST   A1,,[RETURN]
            CDR     A1,TST,[CALL (EVCAR)]
            POP     A3
            XCONS   NIL,A1
            PUSH    A1
EVLIS1:     FLIST   A3,,[JUMP (EVLIS2)]
            PUSH    A1
            CDR     A3,TST
            CAR     A3,A1,[CALL (EVALA1)]
```

```
                POP     A3
                XCONS   NIL,A1
                POP     A2
                SCDR    A1,A2,[JUMP (EVLIS1)]
        EVLIS2: POP     A1,,[RETURN]

                                                 ; (LIST* e1 ... eN)   FSUBR ;
                                                 ; (EVLIS* l)   1SUBR ;
                ; Fabrique une liste de valeurs de manière récursive ;

                ENTRY   LIST*,FSUBR
                ENTRY   EVLIS*,1SUBR

        EVLIS*:
        LIST*:  FLIST   A1,,[RETURN]             ; il n'y a rien à faire ;
                CDR     A1,TST,[CALL (EVCAR)]    ; évalue l'élément suivant ;
                XTOPST  A1,,[CALL (LIST*)]       ; récurse sur les CDR ;
                CONS    TST,A1,[RETURN]          ; fabrique le doublet ;
```

```
            ; SELF EXIT ;
            ;-----------;

            ; SELF : reapplique la même fonction supposée de type EXPR! ;
            ; traite les enveloppes de type WHERE/ESCAPE ;

            ENTRY   SELF,FSUBR,3

SELF:       STACK   A2                          ; sauve le pointeur courant ;
            MOVE    (@ EVALCH),A3               ; récup la chrono actuelle ;
            MOVE    (@ PBIND),A4                ; récup début de bloc ;
SELF1:      EQ      A4,'0,[JUMP (SELFER)]       ; erreur fatale! ;
            SSTACK  A4                          ; pointe sur nouveau bloc ;
            JUMPX   (TSELF),TST                 ; aiguillage sur type de bloc ;
TSELF:      DATA    (SELFL)                     ; Type 0 : bloc LAMBDA ;
            DATA    (SELFW)                     ; Type 1 : bloc WHERE/ESCAPE ;
            DATA    (SELFF)                     ; Type 2 : bloc LETF ;
            DATA    (SELFS)                     ; Type 3 : bloc CHRONOLOGIE ;
SELFL:      POP     A4                          ; récup la Fval empilée  ;
            NEQ     A3,(@ EVALCH),[JUMP (SELFL1)] ; pas même chrono ;
            SSTACK  A2                          ; revient au début du SELF ;
            PUSH    A4,,[JUMP (EVEXP)]          ; empile la nouvelle Fval ;
SELFL1:     NEQ     TST,'*MARK*,[JUMP (SELFL1)] ; dépile le bloc lambda ;
SELFW:
SELFF:      POP     A4,,[JUMP (SELF1)]          ; récupère le nouveau PBIND ;
SELFS:      POP     A4                          ; nouveau PBIND ;
            POP     A3,,[JUMP (SELF1)]          ; nouvelle chrono ;

SELFER:     SSTACK  A2                          ; repositionne SP ;
            PUSH    '*MARK*                     ; fin des arguments ;
            PUSH    '"Erreur SELF : "
            PUSH    A1,,[JUMP (SERROR)]

            ; EXIT (syn: LESCAPE) sort du dernier bloc ;
            ; de type LAMBDA ou WHERE/ESCAPE ;
            ; permet de forcer une récursion terminale ;
            ; doit traiter les enveloppes de type WHERE/ESCAPE ;

            ENTRY   EXIT,FSUBR,3
            ENTRY   LESCAPE,FSUBR,3

LESCAPE:
EXIT:       MOVE    (@ EVALCH),(@ SAVECH)  ; sauve la chrono actuelle ;
EXIT1:      EQ      '0,(@ PBIND),[JUMP (EXITER)] ; erreur fatale ! ;
            SSTACK  (@ PBIND)                   ; passe au bloc suivant ;
            TOPST   A3                          ; pique le type du bloc ;
            EQ      A3,'0,[JUMP (EXIT3)]        ; c'est un bloc LAMBDA ;
EXIT2:      MOVE    (EXIT1),A3,[JUMP (UNBINP)] ; délie le bloc ;
EXIT3:      NEQ     (@ EVALCH),(@ SAVECH),[JUMP (EXIT2)] ; pas bonne chrono ;
            PUSH    (UNBIND),,[JUMP (PROGN)] ; va exécuter le corps ;

EXITER:     PUSH    '*MARK*                     ; fin des arguments ;
            PUSH    '"Erreur EXIT : "
            PUSH    A1,,[JUMP (SERROR)]
```

```
                ; CHRONOLOGY EXITCHRONOLOGY FINDCHRONOLOGY ;
                ;------------------------------------------;

                ; La fonction CHRONOLOGY ;

                ENTRY   CHRONOLOGY,1SUBR

        CHRONOLOGY:
                FNIL    A1,,[JUMP (CHRON1)]
                MOVE    (@ EVALCH),A1,[RETURN]
        CHRON1: MOVE    A1,(@ EVALCH),[RETURN]

                ; EXITCHRONOLOGY FSUBR ;

                ENTRY   EXITCHRONOLOG,FSUBR

        EXITCHRONOLOG:
                NOP     ,,[CALL (PROGN)]        ; valeur à retourner ;
                MOVE    (@ EVALCH),A2           ; chrono actuelle ;
        EXICH1: EQ      '0,(@ PBIND),[JUMP (EXCHER)] ; erreur fatale ! ;
                SSTACK  (@ PBIND)               ; pointe sur le bloc ;
                MOVE    (EXICH2),A3,[JUMP (UNBINP)] ; délie le bloc ;
        EXICH2: EQ      A2,(@ EVALCH),[JUMP (EXICH1)] ; ca a pas bougé ;
                NOP     ,,[RETURN]              ; c'est OK ;

        EXCHER: PUSH    '*MARK*                 ; fin des arguments ;
                PUSH    '"Erreur EXITCHRONOLOGY : "
                PUSH    A1,,[JUMP (SERROR)]

                ; FINDCHRONOLOGY (valeur chrono) ;

                ENTRY   FINDCHRONOLOG,FSUBR

        FINDCHRONOLOG:
                CDR     A1,TST,[CALL (EVCAR)] ; sauve le PROGN ;
                XTOPST  A1,,[CALL (PROGN)]      ; évalue le corps ;
                POP     A2                      ; A2 ← le numéro, A1 la valeur ;
        FINCH1: EQ      '0,(@ PBIND),[JUMP (FICHER)] ; erreur fatale ! ;
                SSTACK  (@ PBIND)               ; pointe sur le bloc ;
                MOVE    (FINCH2),A3,[JUMP (UNBINP)] ; délie un bloc ;
        FINCH2: NEQ     A2,(@ EVALCH),[JUMP (FINCH1)] ; c'est pas cui-la ;
                NOP     ,,[RETURN]              ; c'est cui-la ;

        FICHER: PUSH    '*MARK*                 ; fin des arguments ;
                PUSH    '"Erreur FINDCHRONOLOGY : "
                PUSH    A1,,[JUMP (SERROR)]
```

```
          ; La macro LET ;
          ;--------------;

          ; LET autre forme d'une lambda ;
          ; macro-génère la MACRO ;
          ; (LET () e1 ... eN) ;
          ; (LET (var val) e1 ... eN) ;
          ; (LET ((var val) ... (var val)) e1 ... eN) ;

          ENTRY   LET,FSUBR,2

LET:      CDR     A1,TST                  ; sauve le corps ;
          CAR     A1,A1                   ; A1 ← (var val) ou l'autre forme ;
          MOVE    NIL,A2
          XCONS   NIL,A2                  ; A2 ← (NIL . NIL) ;
          PUSH    A2                      ; A2 ← liste des variables ;
          MOVE    NIL,A3
          XCONS   NIL,A3                  ; A3 ← (NIL . NIL) ;
          PUSH    A3                      ; A3 ← liste des valeurs ;
LET2:     TNIL    A1,,[JUMP (LET8)]       ; Il n'y a plus de variables ;
          CAR     A1,A4
          TLIST   A4,,[JUMP (LET4)]       ; Il y a plusieurs variables ;
          XCONS   NIL,A4                  ; fabrique la liste des variables ;
          SCDR    A4,A2
          CDR     A1,A1                   ; A1 ← (val) ;
          CAR     A1,A4                   ; A4 ← val ;
          XCONS   NIL,A4                  ; A4 ← (VAL) ?!?!?!?!? ;
          SCDR    A4,A3,[JUMP (LET8)]     ; Fabrique la liste des valeurs ;
LET4:     CDR     A1,TST                  ; sauve le reste des couples ;
          CAR     A4,A1                   ; A1 ← var ;
          XCONS   NIL,A1                  ; A1 ← (var) ;
          SCDR    A1,A2                   ; fabrique liste des variables ;
          MOVE    A1,A2                   ; nouvelle fin de liste ;
          CDR     A4,A4                   ; A4 ← (val) ;
          CAR     A4,A1                   ; A1 ← val ;
          XCONS   NIL,A1                  ; A1 ← (val) ?!?!?!?!? ;
          SCDR    A1,A3                   ; fabrique liste des valeurs ;
          MOVE    A1,A3                   ; nouvelle fin de liste ;
          POP     A1,,[JUMP (LET2)]       ; pour le reste des couples ;
LET8:     CDR     TST,A3                  ; A1 ← (lval) ;
          CDR     TST,A2                  ; A2 ← (lvar) ;
          XCONS   TST,A2                  ; A2 ← ((lvar) corps) ;
          CONS    'LAMBDA,A2              ; A2 ← (λ (lvar) corps) ;
          MOVE    (@ FORME),A1            ; pour le RPLACB final ;
          SCAR    A2,A1                   ; (CAR forme) ← (λ (lvar) corps) ;
          SCDR    A3,A1,[JUMP (EVALA1)]   ; (CDR forme) ← (lval) ;
```

```
                ; Fonctions de définition dynamique  WHERE/ESCAPE ;
                ;————————————————————————————————————————————————;

                ; fabriquent un bloc de contrôle de la forme ;

                ;                         [     UNBIND:       ] ;
                ;       PBIND  →          [        1          ] ;
                ;                         [    ancien PBIND   ] ;
                ;                         [ nom de la fonction ] ;
                ;                         [   ancienne F-VAL  ] ;
                ;                         [    ancien F-TYP   ] ;
                ;                         --------------------- ;

                ENTRY   WHERE,FSUBR,2
                ENTRY   ESCAPE,FSUBR,2

        WHERE:  CAR     A1,A2                        ; A2 ← la nouvelle définition ;
                CDR     A1,TST                       ; sauve le corps à exécuter ;
                CAR     A2,TST                       ; sauve le nom de la fonction ;
                CDR     A2,A2                        ; A3 ← nouvelle fonction ;
                CAR     A2,A1                        ; A1 ← F-type ;
                TNUMB   A1,,[JUMP (WHERE1)]          ; forme normale ;
                NOP     ,,[CALL (EVAL)]              ; évalue la nouvelle F-val ;
                MOVE    '7,A4,[JUMP (WHERE2)]        ; F-typ ← EXPR ;
        WHERE1: PUSH    A1                           ; sauve le F-typ ;
                CDR     A2,A1,[CALL (EVCAR)]         ; évalue la F-val ;
                POP     A4                           ; A4 ← le F-typ ;
        WHERE2: MOVE    A1,A3                        ; A3 ← la F-val ;
                POP     A2                           ; A2 ← le nom ;
                POP     A1                           ; A1 ← le corps ;

        WHERE3:                                      ; Liaison dynamique avec : ;
                                                     ;     A1 ← le corps à exécuter ;
                                                     ;     A2 ← le nom de la fonction ;
                                                     ;     A3 ← la nouvelle F-val ;
                                                     ;     A4 ← le nouvel F-typ ;
                FTYP    A2,TST                       ; sauve l'ancien F-TYP ;
                FVAL    A2,TST                       ; sauve l'ancienne F-VAL ;
                PUSH    A2                           ; sauve le nom de la fonction ;
                SFTYP   A4,A2                        ; force le nouveau type ;
                SFVAL   A3,A2                        ; force la nouvelle F-VAL ;
                PUSH    (@ PBIND)                ; sauve l'adresse du bloc precedent ;
                PUSH    '1                           ; type du bloc ;
                STACK   (@ PBIND)                    ; actualise le nouveau PBIND ;
                PUSH    (UNBIND),,[JUMP (PROGN)]     ; prépare le retour ;
                                                     ; et exécute le corps ;


        ESCAPE: MOVE    '10,A4                       ; F-TYP de type ESCAPE ;
                CAR     A1,A2                        ; nom de la fonction ;
                CDR     A1,A1                        ; corps à exécuter ;
                MOVE    A2,A3,[JUMP (WHERE3)]        ; F-VAL = le nom ;

```

```
              ; Variable-fonctions : LETF ;
              ;----------------------------;

              ; (LETF (varfnt s) el ... eN)   FSUBR ;

              ; fabriquent un bloc de contrôle de la forme ;

              ;                          [       UNBIND:       ] ;
              ;      PBIND  →            [          2          ] ;
              ;                          [     ancien PBIND    ] ;
              ;                          [ nom de la fonction  ] ;
              ;                          [   ancienne valeur   ] ;
              ;                          ----------------------- ;

              ENTRY   LETF,FSUBR,3

LETF:         CAR     A1,A2                      ; A2 ← (fnt newval)  ;
              CDR     A2,TST                     ; pile ← (newval) ;
              CDR     A1,TST                     ; pile ← corps ;
              CAR     A2,TST                     ; pile ← fnt ;
              CAR     A2,A2                      ; A2 ← fnt ;
              MOVE    NIL,A1,[CALL (EVALFU)]     ; appel : (FNT) sans arg ;
              XCONS   NIL,A1                     ; fabrique (oldval) ;
              POP     A2                         ; A2 ← fnt ;
              POP     A3                         ; A3 ← corps ;
              XTOPST  A1                         ; A1 ← (newval) ;
              PUSH    A2                         ; pile ← fnt ;
              PUSH    A3,,[CALL (EVALFU)]        ; sauve le corps ;
              POP     A1                         ; A1 ← le corps ;
              PUSH    (@ PBIND)                  ; sauve le vieux PBIND ;
              PUSH    '2                         ; type du bloc ;
              STACK   (@ PBIND)                  ; positionne le nouveau PBIND ;
              PUSH    (UNBIND),,[JUMP (PROGN)]   ; prépare le UNBIND et  ;
                                                 ; exécute le corps ;
```

```
                ; Fonctions de définition statique ;
                ;---------------------------------;

RDEF:   PUSH    A1                              ; sauve tout l'appel ;
        CAR     A1,A1                           ; A1 ← le nom de la fonction ;
        FVAL    A1,A2
        XCONS   NIL,A2                          ; fabrique (FVAL) ;
        FTYP    A1,A3
        CONS    A3,A2                           ; A2 ← (FTYP FVAL) ;
        MOVE    'INTERNAL,A3,[CALL (ADDPROP)]
                                                  ; sauve l'ancienne définition ;
        POP     A1                              ; récupère tout l'appel ;
                                                ; DEF doit suivre ... ;
DEF:    CDR     A1,A2                           ; A2 ← ((lvar) corps ) ;
        CAR     A1,A1                           ; A1 ← nom de la fonction ;
        SFTYP   A3,A1                           ; force le nouveau FTYP ;
        SFVAL   A2,A1,[RETURN]                  ; force la nouvelle FVAL ;

        ; DE DF DM ;
        ; détruisent l'ancien couple FTYP/FVAL ;

        ENTRY   DE,FSUBR,1
        ENTRY   DF,FSUBR,1
        ENTRY   DM,FSUBR,1

DE:     MOVE    '7,A3,[JUMP (DEF)]              ; FTYP = 7 : EXPR ;
DF:     MOVE    '8,A3,[JUMP (DEF)]              ; FTYP = 8 : FEXPR ;
DM:     MOVE    '9,A3,[JUMP (DEF)]              ; FTYP = 9 : MACRO ;

        ; RDE RDF RDM ;
        ; sauvent l'ancien couple FTYP/FVAL ;

        ENTRY   RDE,FSUBR,1
        ENTRY   RDF,FSUBR,1
        ENTRY   RDM,FSUBR,1

RDE:    MOVE    '7,A3,[JUMP (RDEF)]             ; FTYP = 7 : EXPR ;
        MOVE    '8,A3,[JUMP (RDEF)]             ; FTYP = 8 : FEXPR ;
        MOVE    '9,A3,[JUMP (RDEF)]             ; FTYP = 9 : MACRO ;

        ; (REVERT at) ;
        ; remet l'ancienne définition de at ;

        ENTRY   REVERT,1SUBR

REVERT: PUSH    A1                              ; sauve le nom de la fonction ;
        MOVE    'INTERNAL,A2,[CALL (GET)]       ; indicateur de sauvetage ;
        POP     A3                              ; récupère le nom ;
        TNIL    A1,,[RETURN]                    ; il n'y avait pas de définition ;
        CAR     A1,A4                           ; A4 ← l'ancien FTYP ;
        SFTYP   A4,A3                           ; restauré ;
        CDR     A1,A4
        CAR     A4,A4                           ; A4 ← l'ancienne FVAL ;
        SFVAL   A4,A3,[JUMP (REMPROP)]
                                        ; restaurée et enlève l'ancienne def ;
```

```
                ; Fonctions d'accès aux attributs des atomes ;
                ;------------------------------------------;

                ; Accès aux attributs : CVAL PLIST FVAL FTYP ;
                ; Toutes ces fonction sont en GET/SET ;

                ENTRY   CVAL,1SUBR
                ENTRY   PLIST,2SUBR
                ENTRY   FVAL,2SUBR
                ENTRY   FTYPE,2SUBR

CVAL:           CVAL    A1,A1,[RETURN]              ; ramène la CVAL courante ;

PLIST:          FATOM   A1,,[JUMP (FALSE)]          ; il faut un atome litteral ;
                TNIL    A1,,[RETURN]                ; différent de NIL ;
                TNIL    A2,,[JUMP (PLIST1)]         ; pas de 2ème argument ;
                SPLIST  A2,A1                       ; force la nouvelle PLIST ;
PLIST1:         PLIST   A1,A1,[RETURN]              ; ramène la PLIST courante ;

FVAL:           TNIL    A2,,[JUMP (FVAL1)]          ; pas de 2ème arg ;
                SFVAL   A2,A1                       ; force la nouvelle FVAL ;
FVAL1:          FVAL    A1,A1,[RETURN]              ; ramène la FVAL courante ;

FTYPE:          TNIL    A2,,[JUMP (FTYP1)]          ; pas de 2ème arg ;
                SFTYP   A2,A1                       ; force le nouveau FTYP ;
FTYP1:          FTYP    A1,A1,[RETURN]              ; ramène le FTYP courant ;

                ; (SYNONYM at1 at2) ;

                ENTRY   SYNONYM,2SUBR

SYNONYM:FVAL    A2,A3                               ; A3 ← FVAL de at2 ;
                SFVAL   A3,A1                       ; force la nouvelle FVAL de at1 ;
                FTYP    A2,A3                       ; A3 ← FTYP de at2 ;
                SFTYP   A3,A1                       ; force le nouveau FTYP de at1 ;
                PTYP    A2,A3                       ; A3 ← PTYP de at2 ;
                SPTYP   A3,A1,[RETURN]              ; force le nouveau PTYP de at1 ;
```

```
        ; Fonctions de contrôle ;
        ;-----------------------;

        ; IF : FSUBR. La fonction conditionnelle la plus simple ;
        ; et sa soeur IFN qui possède la condition inversée ;
        ; permet de traiter les recursions terminales ;

        ENTRY   IF,FSUBR,2
        ENTRY   IFN,FSUBR,2

IF:     PUSH    A1,,[CALL (EVCAR)]      ; evaluate the predicate ;
        CDR     TST,A2
        CDR     A2,A3
        TNIL    A1,,[JUMP (PROGNA3)]    ; else clauses ;
        CAR     A2,A1,[JUMP (EVALA1)]   ; then clause ;

IFN:    PUSH    A1,,[CALL (EVCAR)]      ; evaluate the predicate ;
        CDR     TST,A2
        CDR     A2,A3
        FNIL    A1,,[JUMP (PROGNA3)]    ; else clauses ;
        CAR     A2,A1,[JUMP (EVALA1)]   ; then clause ;

        ; COND : FSUBR. La fonction conditionnelle la plus connue! ;
        ; permet de traiter les recursions terminales ;

        ENTRY   COND,FSUBR,5

COND:   MOVE    A1,A2
COND1:  FLIST   A2,,[RETURN]            ; ya plus de clauses ;
        CDR     A2,TST                  ; sauve le reste des clauses ;
        CAR     A2,A1                   ; A1 ← clause suivante ;
        CDR     A1,TST,[CALL (EVCAR)]   ; évalue le prédicat ;
        POP     A3                      ; A3 ← le corps de la clause ;
        POP     A2                      ; A2 ← le reste des clauses ;
        TNIL    A1,,[JUMP (COND1)]      ; le prédicat à ramener NIL ;
        FNIL    A3,,[JUMP (PROGNA3)]    ; évalue le corps de la clause ;
        NOP     ,,[RETURN]              ; en cas de clause vide ;

        ; OR AND : FSUBR, les connecteurs logiques ;
        ; permet de traiter les recursions terminales ;

        ENTRY   OR,FSUBR,3
        ENTRY   AND,FSUBR,3

OR:     CDR     A1,A2
        FLIST   A2,,[JUMP (EVCAR)]      ; le dernier élément ;
        PUSH    A2,,[CALL (EVCAR)]
        FNIL    A1,,[JUMP (PRET)]
        POP     A1,,[JUMP (OR)]

AND:    FLIST   A1,,[JUMP (TRUE)]       ; (AND) -> T ;
AND1:   CDR     A1,A2
        FLIST   A2,,[JUMP (EVCAR)]      ; le dernier élément ;
        PUSH    A2,,[CALL (EVCAR)]
        TNIL    A1,,[JUMP (PRET)]
        POP     A1,,[JUMP (AND1)]       ;;

PRET:   POP     A2,,[RETURN]            ; dépile et retourne ;

        ; WHILE : FSUBR et UNTIL : FSUBR ;
        ; ne permet évidemment pas de traiter les récursions terminales ;
```

```
        ; car dans un WHILE il n'y a rien  en position terminale ;

        ENTRY   WHILE,FSUBR,2
        ENTRY   UNTIL,FSUBR,2

WHILE:  PUSH    A1,,[JUMP (WHILE2)]     ; empile toute l'expression ;
WHILE1: TOPST   A1                      ; récupère l'expression entière ;
        CDR     A1,A1,[CALL (PROGN)]    ; pour évaluer le corps ;
WHILE2: TOPST   A1,,[CALL (EVCAR)]      ; évalue le  test ;
        FNIL    A1,,[JUMP (WHILE1)]     ; il faut re-évaluer le corps ;
        POP     A2,,[RETURN]            ; nettoie la pile et rentre ;

UNTIL:  PUSH    A1,,[JUMP (UNTIL2)]     ; empile toute l'expression ;
UNTIL1: TOPST   A1                      ; récupère l'expression entière ;
        CDR     A1,A1,[CALL (PROGN)]    ; pour évaluer le corps ;
UNTIL2: TOPST   A1,,[CALL (EVCAR)]      ; évalue le  test ;
        TNIL    A1,,[JUMP (UNTIL1)]     ; il faut re-évaluer le corps ;
        POP     A2,,[RETURN]            ; nettoie la pile et rentre ;

        ; SELECTQ ;

        ENTRY   SELECTQ,FSUBR,4

SELECTQ:CDR     A1,TST,[CALL (EVCAR)]   ; évalue le selecteur ;
        POP     A3,,[JUMP (SELEC4)]     ; récupère les clauses ;
SELEC2: EQ      A1,A4,[JUMP (PROGNA3)]; c'est cette clause ;
SELEC3: MOVE    A2,A3                 ; A3 ← l'ancien reste des clauses ;
SELEC4: CDR     A3,A2                   ; A2 ← le reste des clauses ;
        CAR     A3,A3                   ; A3 ← la clause suivante ;
        FLIST   A2,,[JUMP (PROGNA3)]    ; c'est la clause fausse ;
        CAR     A3,A4                   ; A4 ← le selecteur ;
        CDR     A3,A3                   ; A3 ← le corps de la clause ;
        FLIST   A4,,[JUMP (SELEC2)]     ; c'est un sélecteur simple ;
        PUSH    A1                      ; sauve le sélecteur ;
        PUSH    A2                      ; sauve le reste des clauses ;
        PUSH    A3                      ; sauve le reste de la clause ;
        MOVE    A4,A2,[CALL (MEMBER)]   ; teste le sélecteur complexe ;
        POP     A3                      ; récupère le reste de la clause ;
        POP     A2                      ; récupère le reste des clauses ;
        XTOPST  A1                      ; A1 ← le sélecteur ;
        TNIL    TST,,[JUMP (SELEC3)]    ; c'est pas la bonne clause ;
        NOP     ,,[JUMP (PROGNA3)]      ; c'est celle-là ;
```

```
                ; Fonctionnelles ;
                ;----------------;

                ; (MAPC fnt larg1 ... largN) ;
                ; (MAP fnt larg1 ... largN) ;

                ENTRY   MAPC,NSUBR,2
                ENTRY   MAP,NSUBR,2

        MAPC:   CAR     A1,TST                      ; empile la fonction ;
                PUSH    '*MARK*,,[JUMP (MAPC11)]    ; marque de fin des liste-args ;
        MAPC1:  CAR     A1,TST                      ; empile la liste-arg suivante ;
        MAPC11: CDR     A1,A1                       ; avance dans les liste-args ;
                TLIST   A1,,[JUMP (MAPC1)]          ; il en reste ;
        MAPC2:  STACK   (@ SAVEP)                   ; sauve le haut de la pile ;
                MOVE    NIL,A2                      ; liste-arg pour APPLY ;
                MOVE    NIL,A3,[JUMP (MAPC6)]
                                                  ; indic toutes les listes sont vides ;
        MAPC3:  TLIST   A4,,[JUMP (MAPC4)]          ; la liste-arg est vide ;
                CONS    A4,A2,[JUMP (MAPC5)]        ; élément simple ;
        MAPC4:  MOVE    A4,A3                       ; indic vrai ;
                CAR     A4,A1                       ; CAR pour MAPC ;
                CONS    A1,A2                       ; larg pour APPLY ;
                CDR     A4,A4                       ; avance dans liste-arg ;
        MAPC5:  XTOPST  A4                          ; remet liste-arg à sa place ;
                POP     A4                          ; on saute la liste-arg traitée ;
        MAPC6:  TOPST   A4                          ; liste-arg suivante ;
                NEQ     A4,'*MARK*,[JUMP (MAPC3)]   ; il en reste ;
                POP     A4                          ; enlève la marque ;
                POP     A1                          ; récupère la FVAL ;
                TNIL    A3,,[JUMP (FALSE)]          ; c'est fini ;
                SSTACK  (@ SAVEP),,[CALL (APPLY)]   ; remet la pile en ordre ;
                NOP     ,,[JUMP (MAPC2)]

        MAP:    CAR     A1,TST                      ; empile la fonction ;
                PUSH    '*MARK*,,[JUMP (MAP11)]     ; marque de fin des liste-args ;
        MAP1:   CAR     A1,TST                      ; empile la liste-arg suivante ;
        MAP11:  CDR     A1,A1                       ; avance dans les liste-args ;
                TLIST   A1,,[JUMP (MAP1)]           ; il en reste ;
        MAP2:   STACK   (@ SAVEP)                   ; sauve le haut de la pile ;
                MOVE    NIL,A2                      ; liste-arg pour APPLY ;
                MOVE    NIL,A3,[JUMP (MAP6)]
                                                  ; indic toutes les listes sont vides ;
        MAP3:   TLIST   A4,,[JUMP (MAP4)]           ; la liste-arg est vide ;
                CONS    A4,A2,[JUMP (MAP5)]         ; atome simple ;
        MAP4:   MOVE    A4,A3                       ; indic vrai ;
                CONS    A4,A2                       ; larg pour APPLY ;
                CDR     A4,A4                       ; avance dans liste-arg ;
        MAP5:   XTOPST  A4                          ; remet liste-arg à sa place ;
                POP     A4                          ; on saute la liste-arg traitée ;
        MAP6:   TOPST   A4                          ; liste-arg suivante ;
                NEQ     A4,'*MARK*,[JUMP (MAP3)]    ; il en reste ;
                POP     A4                          ; enlève la marque ;
                POP     A1                          ; récupère la FVAL ;
                TNIL    A3,,[JUMP (FALSE)]          ; c'est fini ;
                SSTACK  (@ SAVEP),,[CALL (APPLY)]   ; remet la pile en ordre ;
                NOP     ,,[JUMP (MAP2)]
```

```
            ; Prédicats ;
            ;-----------;

            ; NULL NOT ATOM NUMBP LISTP BOUNDP : 1SUBR ;

            ENTRY  NULL,1SUBR
            ENTRY  NOT,1SUBR
            ENTRY  ATOM,1SUBR
            ENTRY  NUMBP,1SUBR
            ENTRY  LISTP,1SUBR
            ENTRY  BOUNDP,1SUBR

NULL:
NOT:        TNIL   A1,,[JUMP (TRUE)]
            MOVE   NIL,A1,[RETURN]

ATOM:       FLIST  A1,,[JUMP (TRUE)]
            MOVE   NIL,A1,[RETURN]

NUMBP:      TNUMB  A1,,[RETURN]
            MOVE   NIL,A1,[RETURN]

LISTP:      TLIST  A1,,[JUMP (TRUE)]
            MOVE   NIL,A1,[RETURN]

BOUNDP:     CVAL   A1,A2
            NEQ    A2,'~UNDEF,[JUMP (TRUE)]
            MOVE   NIL,A1,[RETURN]

            ; EQ NEQ : 2SUBR ;

            ENTRY  EQ,2SUBR
            ENTRY  NEQ,2SUBR

EQ:         EQ     A1,A2,[JUMP (TRUE)]
            MOVE   NIL,A1,[RETURN]

NEQ:        NEQ    A1,A2,[JUMP (TRUE)]
            MOVE   NIL,A1,[RETURN]

            ; EQUAL NEQUAL : 2SUBR ;

            ENTRY  EQUAL,2SUBR
            ENTRY  NEQUAL,2SUBR

NEQUAL:     PUSH   (NOT)
EQUAL:      STACK  (@ SAVEP),,[CALL (EQUAL2)]
                                ; sauve le SP pour les retours rapides ;
            MOVE   'T,A1,[RETURN]
EQUAL1:     FLIST  A2,,[JUMP (NAN)]
            CDR    A1,TST                      ; sauve les 2 CDR ;
            CAR    A1,A1
            CDR    A2,TST
            CAR    A2,A2,[CALL (EQUAL2)]       ; récursion sur les CAR ;
            POP    A2
            POP    A1
EQUAL2:     TLIST  A1,,[JUMP (EQUAL1)]         ; itération sur les CDR ;
            EQ     A1,A2,[RETURN]
NAN:        SSTACK (@ SAVEP)                   ; ramène la pile a sa juste valeur ;
FALSE:      MOVE   NIL,A1,[RETURN]             ; retour FAUX ;

TRUE:       MOVE   'T,A1,[RETURN]              ; retour VRAI ;
```

1276

```
                ; Recherches ;
                ;------------;

                ; CAR CDR C..R : 1SUBR ;

                ENTRY   QUOTE,FSUBR
                ENTRY   CAR,1SUBR
                ENTRY   CDR,1SUBR
                ENTRY   CAAR,1SUBR
                ENTRY   CADR,1SUBR
                ENTRY   CDAR,1SUBR
                ENTRY   CDDR,1SUBR
                ENTRY   CAAAR,1SUBR
                ENTRY   CAADR,1SUBR
                ENTRY   CADAR,1SUBR
                ENTRY   CADDR,1SUBR
                ENTRY   CDAAR,1SUBR
                ENTRY   CDADR,1SUBR
                ENTRY   CDDAR,1SUBR
                ENTRY   CDDDR,1SUBR

        CAAAR:  CAR     A1,A1
        CAAR:   CAR     A1,A1
        CAR:    CAR     A1,A1,[RETURN]

        CAADR:  CDR     A1,A1
                CAR     A1,A1
                CAR     A1,A1,[RETURN]

        CADAR:  CAR     A1,A1
        CADR:   CDR     A1,A1
                CAR     A1,A1,[RETURN]

        CADDR:  CDR     A1,A1
                CDR     A1,A1
                CAR     A1,A1,[RETURN]

        CDAAR:  CAR     A1,A1
        CDAR:   CAR     A1,A1
                CDR     A1,A1,[RETURN]

        CDADR:  CDR     A1,A1
                CAR     A1,A1
                CDR     A1,A1,[RETURN]

        CDDAR:  CAR     A1,A1
                CDR     A1,A1
                CDR     A1,A1,[RETURN]

        CDDDR:  CDR     A1,A1
        CDDR:   CDR     A1,A1
        CDR:    CDR     A1,A1,[RETURN]

                ; (MEMQ a l)   2SUBR ;

                ENTRY   MEMQ,2SUBR

        MEMQ1:  CAR     A2,A3                   ; A3 ← l'élément suivant ;
                EQ      A3,A1,[JUMP (MEMQ2)]    ; c'est le sélecteur ? ;
                CDR     A2,A2                   ; nan : continue ;
        MEMQ:   TLIST   A2,,[JUMP (MEMQ1)]      ; la liste est vide ;
        MEMQ2:  MOVE    A2,A1,[RETURN]          ; ramène le dernier CDR ;
```

```
                ; (MEMBER s l)   2SUBR ;

                ENTRY   MEMBER,2SUBR

MEMB1:          PUSH    A2                          ; sauve la liste en entier ;
                PUSH    A1                          ; sauve le test ;
                CAR     A2,A2,[CALL (EQUAL)]        ; élément suivant de la liste ;
                FNIL    A1,,[JUMP (MEMB3)]          ; c'est celui-là ;
                POP     A1                          ; récupère le test ;
                CDR     TST,A2                      ; suite de la liste ;
MEMBER:         TLIST   A2,,[JUMP (MEMB1)]          ; la liste est vide ? ;
                MOVE    A2,A1,[RETURN]              ; ramène le dernier CDR ;
MEMB3:          POP     A1                          ; récup le test ;
                POP     A1,,[RETURN]                ; retourne la liste en entier ;

                ; (LAST l)  1SUBR ;
                ; retourne le dernier doublet d'une liste ;
                ; 2 instructions par élément ! ;

                ENTRY   LAST,1SUBR

LAST:           FLIST   A1,,[RETURN]                ; pas d'argument ;
LAST1:          CDR     A1,A2                       ; avance dans la liste ;
                FLIST   A2,,[RETURN]                ; A1 est le dernier doublet ;
                CDR     A2,A1                       ; avance dans la liste ;
                TLIST   A1,,[JUMP (LAST1)]          ; elle est encore longue ;
                MOVE    A2,A1,[RETURN]              ; retourne le dernier doublet ;

                ; (NTH n l)   2SUBR ;
                ; (CNTH n l)  2SUBR ;

                ENTRY   NTH,2SUBR
                ENTRY   CNTH,2SUBR

CNTH:           PUSH    (CAR),,[JUMP (NTH)]         ; pour le retour de CNTH ;
NTH1:           CDR     A2,A2                       ; avance dans la liste ;
                FLIST   A2,,[JUMP (FALSE)]          ; elle est terminee = NIL ;
NTH:            SUB     '1,A1                       ; decremente le compteur ;
                GT      A1,'0,[JUMP (NTH1)]         ; il faut encore avancer ;
                MOVE    A2,A1,[RETURN]              ; ramène la liste en l'etat ;

                ; (LENGTH l)   1SUBR ;

                ENTRY   LENGTH,1SUBR

LENGTH:         MOVE    '0,A2,[JUMP (LENGT2)]       ; raz le compteur ;
LENGT1:         CDR     A1,A1                       ; avance dans la liste ;
                ADD     '1,A2                       ; et compte ;
LENGT2:         TLIST   A1,,[JUMP (LENGT1)]         ; la liste continue ;
                MOVE    A2,A1,[RETURN]              ; retourne la taille de la liste ;
```

```
                ; Création d'objets ;
                ;-------------------;

                ; (CONS s1 s2)   2SUBR ;
                ; (XCONS s1 s2)   2SUBR ;
                ; (NCONS s1)   1SUBR ;

                ENTRY   CONS,2SUBR
                ENTRY   XCONS,2SUBR
                ENTRY   NCONS,1SUBR

        CONS:   XCONS   A2,A1,[RETURN]

        XCONS:  CONS    A2,A1,[RETURN]

        NCONS:  XCONS   NIL,A1,[RETURN]

                ; (MCONS e1 ... eN)   FSUBR ;

                ENTRY   MCONS,FSUBR,3

        MCONS:  FLIST   A1,,[RETURN]            ; (MCONS) = NIL ;
        MCONS1: CDR     A1,A2
                FLIST   A2,,[JUMP (EVCAR)]      ; evaluation du dernier ;
                PUSH    A2,,[CALL (EVCAR)]      ; evaluation du suivant ;
                XTOPST  A1,,[CALL (MCONS1)]     ; ca recurse ;
                CONS    TST,A1,[RETURN]

                ; (KWOTE s)   1SUBR ;

                ENTRY   KWOTE,1SUBR

        KWOTE:  XCONS   NIL,A1                  ; (arg) ;
                CONS    'QUOTE,A1,[RETURN]      ; (QUOTE (arg)) ;

                ; (REVERSE l1 l2)   2SUBR ;

                ENTRY   REVERSE,2SUBR

        REV1:   CAR     A1,A3
                CDR     A1,A1
                CONS    A3,A2
        REVERSE:TLIST   A1,,[JUMP (REV1)]
                MOVE    A2,A1,[RETURN]

                ; (APPEND l1 l2)   2SUBR ;
                ; (APPEND1 l a)    2SUBR ;

                ENTRY   APPEND,2SUBR
                ENTRY   APPEND1,2SUBR

        APPEND1:XCONS   NIL,A2                  ; Cons le 2ème argument ;
                                                ; APPEND doit suivre ... ;
        APPEND: MOVE    NIL,A3                  ; Fabrique le 1er doublet ;
                XCONS   NIL,A3
                PUSH    A3,,[JUMP (APPEN3)]
        APPEN2: CAR     A1,A4
                XCONS   NIL,A4
                SCDR    A4,A3                   ; rajoute le nouvel élément ;
                MOVE    A4,A3                   ; nouvelle fin de liste ;
                CDR     A1,A1                   ; avance dans la 1ère liste ;
        APPEN3: TLIST   A1,,[JUMP (APPEN2)]     ; il en reste ;
```

```
                FLIST  A2,,[JUMP (APPEN4)]      ; s'il n'y a pas de 2ème arg ;
                SCDR   A2,A3                    ; accroche le 2ème argument ;
        APPEN4: CDR    TST,A1,[RETURN]      ; ramène le CDR de la liste empilée ;

                ; (DELQ l a)   2SUBR ;
                ; actuellement récursif. ;

                ENTRY  DELQ,2SUBR

        DELQ3:  CDR    A1,A1
        DELQ:   TNIL   A1,,[RETURN]             ; la liste est vide ;
                CAR    A1,A3                    ; A3 ← élément suivant de l ;
                EQ     A3,A2,[JUMP (DELQ3)]     ; si c'est à enlever ;
                PUSH   A3                       ; sauve l'élément à CONSER ;
                CDR    A1,A1,[CALL (DELQ)]  ; récurse sur le reste de la liste ;
                CONS   TST,A1,[RETURN]          ; et CONS récursif en retour ;

                ; (COPY l)   1SUBR ;
                ; actuellement récursif. A itérativer! ;

                ENTRY  COPY,1SUBR

        COPY:   FLIST  A1,,[RETURN]             ; les atomes ne changent pas ;
                CDR    A1,TST                   ; sauve le CDR ;
                CAR    A1,A1,[CALL (COPY)]      ; récurse sur les CARs ;
                XTOPST A1,,[CALL (COPY)]        ; ET SUR LES CDRs! ;
                CONS   TST,A1,[RETURN]          ; et construit enfin le doublet ;

                ; (SUBST new old e)   3SUBR ;
                ; Petit SUBST avec EQ ;

                ENTRY  SUBST,3SUBR

        SUBST:  FATOM  A3,,[JUMP (SUBST1)]      ; SUBST est parfait ;
                EQ     A3,A2,[RETURN]
                MOVE   A3,A1,[RETURN]
        SUBST1: PUSH   A1
                CDR    A3,TST
                CAR    A3,A3,[CALL (SUBST)]
                POP    A3
                XTOPST A1,,[CALL (SUBST)]
                CONS   TST,A1,[RETURN]

                ; (SUBST* new old e)   3SUBR ;
                ; Le gros actuellement récursif. A itérativer ;

                ENTRY  SUBST*,3SUBR

        SUBST*: NEQ    A1,A2,[JUMP (SUBST1*)]   ; SUBST est parfait ;
                MOVE   A3,A1,[JUMP (COPY)]      ; il vaut mieux utiliser COPY ;
        SUBST1*:PUSH   A1                       ; sauve new ;
                MOVE   A3,A1                    ; change e et new ;
                POP    A3                       ; A1 ← e, A2 ← old, A3 ← new ;
        SUBST2*:PUSH   A1                       ; sauve e ;
                PUSH   A2                       ; sauve old ;
                PUSH   A3,,[CALL (EQUAL)]       ; sauve new et test e::old ;
                POP    A3                       ; restaure new ;
                POP    A2                       ; restaure old ;
                TNIL   A1,,[JUMP (SUBST3*)]     ; ce n'est pas le même ;
                POP    A1                       ; enlève l'ancien e ;
                MOVE   A3,A1,[RETURN]           ; et retourne new ;
        SUBST3*:POP    A1                       ; restaure e ;
                FLIST  A1,,[RETURN]             ; l'arbre est terminé ;
```

```
        CDR     A1,TST                      ; sauve le CDR ;
        CAR     A1,A1,[CALL (SUBST2*)]      ; récurse sur les CARs ;
        XTOPST  A1,,[CALL (SUBST2*)]        ; RESURSE SUR LES CDRs! ;
        CONS    TST,A1,[RETURN]             ; et construit le doublet ;
```

```
          ; Modification d'objets ;
          ;----------------------;

          ; RPLACA RPLACD RPLACB : 2SUBR ;
          ; ne sont definies que sur les listes ;

          ENTRY   RPLACA,2SUBR
          ENTRY   RPLACD,2SUBR
          ENTRY   RPLACB,2SUBR

RPLACA:   SCAR    A2,A1,[RETURN]
RPLACD:   SCDR    A2,A1,[RETURN]
RPLACB:   CAR     A2,A3                    ; pour le RPLACA ;
          CDR     A2,A4                    ; pour le RPLACD ;
          SCAR    A3,A1                    ; change le CAR ;
          SCDR    A4,A1,[RETURN]           ; change le CDR ;

          ; SETQ : FSUBR ;

          ENTRY   SETQ,FSUBR

SETQ1:    MOVE    A2,A1
SETQ:     CAR     A1,TST                   ; sauve le nom de la variable ;
          CDR     A1,A1                    ; A1 ← (val var val ...) ;
          CDR     A1,TST,[CALL (EVCAR)]    ; sauve le reste et évalue la val ;
          POP     A2                       ; récupère le reste de la liste ;
          POP     A3                       ; récupère le nom de la variable ;
          SCVAL   A1,A3                    ; force sa CVAL ;
          TLIST   A2,,[JUMP (SETQ1)]       ; il reste encore des couples ;
          NOP     ,,[RETURN]               ; et ramène la dernière valeur ;

          ; SET : 2SUBR ;
          ; n'est definie que sur les variables ;

          ENTRY   SET,2SUBR

SET:      SCVAL   A2,A1                    ; force la nouvelle valeur ;
          MOVE    A2,A1,[RETURN]           ; qui est ramenee en valeur ;

          ; SETQQ : FSUBR ;
          ; n'est definie que sur les variables ;

          ENTRY   SETQQ,FSUBR

SETQQ:    CAR     A1,A2                    ; A2 ← la variable ;
          CDR     A1,A1
          CAR     A1,A1                    ; A1 ← la valeur ;
          SCVAL   A1,A2,[RETURN]           ; forcée et ramenée en valeur ;

          ; FREVERSE : 2SUBR ;

          ENTRY   FREVERSE,2SUBR
          ; 5 intructions par doublets ;

FREV1:    CDR     A1,A3                    ; avance dans la liste ;
          SCDR    A2,A1                    ; change le CDR ;
          MOVE    A1,A2                    ; A2 ← nouvelle liste ;
          MOVE    A3,A1                    ; A1 ← CDR A1 ;
FREVERSE:
          TLIST   A1,,[JUMP (FREV1)]       ; la liste continue ;
          MOVE    A2,A1,[RETURN]           ; ramene la nouvelle liste ;
```

```
          ; FREVERSE* : 2SUBR ;

          ENTRY   FREVERSE*,2SUBR
          ; optimise les transferts : 3 instructions par doublets ;

FREV1*:   CDR     A1,A3                   ; avance dans la liste ;
          SCDR    A2,A1                   ; change le CDR ;
          FLIST   A3,,[RETURN]            ; la liste est terminée ;
          CDR     A3,A2                   ; avance dans la liste ;
          SCDR    A1,A3                   ; change le CDR ;
          FLIST   A2,,[JUMP (FREV9*)]     ; la liste est terminée ;
          CDR     A2,A1                   ; avance dans la liste ;
          SCDR    A3,A2                   ; change le CDR ;
FREVERSE*:
          TLIST   A1,,[JUMP (FREV1)]      ; la liste continue ;
          MOVE    A2,A1,[RETURN]          ; retourne la nouvelle liste ;
FREV9*:   MOVE    A3,A1,[RETURN]          ; retourne la nouvelle liste ;

          ; NEXTL : FSUBR ;

          ENTRY   NEXTL,FSUBR

NEXTL:    CAR     A1,A2                   ; A2 ← le nom de l'atome ;
          CVAL    A2,A3                   ; A3 ← sa C-VAL ;
          CAR     A3,A1                   ; A1 ← le CAR de la C-VAL ;
          CDR     A3,A3                   ; A3 ← le CDr de la C-VAL ;
          SCVAL   A3,A2,[RETURN]          ; qui devient sa nouvelle C-VAL ;

          ; (NEWL at s) FSUBR ;

          ENTRY   NEWL,FSUBR

NEWL:     CAR     A1,TST                  ; sauve le nom de l'atome ;
          CDR     A1,A1,[CALL (EVCAR)]    ; évalue l'expression ;
          POP     A2                      ; récupère le nom de l'atome ;
          CVAL    A2,A3                   ; A3 ← ancienne CVAL de l'atome ;
          XCONS   A3,A1                   ; fabrique la nouvelle valeur ;
          SCVAL   A1,A2,[RETURN]          ; qui est rangée dans la CVAL ;

          ; (NCONC l1 l2)  2SUBR ;
          ; (NCONC1 l1 l2) 2SUBR ;

          ENTRY   NCONC,2SUBR
          ENTRY   NCONC1,2SUBR

NCONC1:   XCONS   NIL,A2                  ; listifie le 2ème argument ;

NCONC:    TLIST   A1,,[JUMP (NCONC2)]     ; NCONC peut démarrer ;
          MOVE    A2,A1,[RETURN]          ; sinon retourne le 2ème argument ;
NCONC2:   PUSH    A1,,[JUMP (NCONC4)]     ; sauve la valeur de retour ;
NCONC3:   MOVE    A3,A1                   ; A1 ← CDR A1 ;
NCONC4:   CDR     A1,A3                   ; avance dans la liste 1er argument ;
          TLIST   A3,,[JUMP (NCONC3)]     ; elle n'est pas terminée ;
          SCDR    A2,A1                   ; force le dernier CDR ;
          POP     A1,,[RETURN]            ; et retourne la liste initiale ;
```

```
        ; Fonctions sur P-LISTES ;
        ;------------------------;

        ; Ne supportent que les Plistes des atomes ;
        ; les indicateurs sont des atomes ;

        ENTRY   GET,2SUBR
        ENTRY   ADDPROP,3SUBR
        ENTRY   PUT,3SUBR
        ENTRY   REMPROP,2SUBR

        ; GET interne teste si l'indic A3 est présent dans la ;
        ;     PLIST de A1. Ramène dans A4 le début de la PLIST de ;
        ;     A1 qui commence à l'indicateur A3 ;

GETI:   TATOM   A1,,[JUMP (GETI1)]      ; c'est un bon atome littéral ;
GETI0:  MOVE    NIL,A4,[RETURN]         ; sinon retourne faux ;
GETI1:  TNIL    A1,,[JUMP (GETI0)]      ; ne pas toucher à NIL ;
        PLIST   A1,A4,[JUMP (GETI3)]    ; A4 ← la PLIST de l'atome ;
GETI2:  CAR     A4,A2                   ; indicateur suivant ;
        EQ      A3,A2,[RETURN]          ; C'est tout bon ;
        CDR     A4,A4                   ; saute l'indicateur ;
        CDR     A4,A4                   ; saute la valeur ;
GETI3:  TLIST   A4,,[JUMP (GETI2)]      ; la PLIST se continue ;
        NOP     ,,[RETURN]              ; elle est vide ;

GET:                                  ; A1 ← la P-liste, A2 ← l'indicateur ;
        MOVE    A2,A3,[CALL (GETI)]     ; recherche de l'indicateur ;
        FLIST   A4,,[JUMP (FALSE)]      ; la recherche a échouée ;
        CDR     A4,A4                   ; pointe sur la (valeur) ;
        CAR     A4,A1,[RETURN]          ; ramène la valeur simple ;

ADDPROP:TNIL    A1,,[RETURN]            ; ne pas toucher à NIL ;
        TLIST   A1,,[RETURN]            ; ce doit être un atome littéral ;
        PLIST   A1,A4                   ; récpère l'ancienne PLIST ;
        CONS    A2,A4                   ; (val . pl) ;
        CONS    A3,A4                   ; (ind val . pl) ;
        SPLIST  A4,A1,[RETURN]          ; force la nouvelle PLIST ;

PUT:    PUSH    A2,,[CALL (GETI)]       ; sauve la PVAL et rech l'indic ;
        POP     A2                      ; récupère la PVAL ;
        FLIST   A4,,[JUMP (ADDPROP)]    ; L'indic n'etait pas present ;
        CDR     A4,A4                   ; pointe sur la (valeur) ;
        SCAR    A2,A4,[RETURN]          ; change la valeur ;

REMPROP:MOVE    A2,A3,[CALL (GETI)]     ; recherche de l'indicateur ;
        FLIST   A4,,[JUMP (FALSE)]      ; la recherche a échouée ;
        CDR     A4,A3                   ; A3 ← ( val indic val ...) ;
        CDR     A3,A3                   ; A3 ← (indic val ... suivants) ;
        CAR     A3,A2                   ; indic ;
        SCAR    A2,A4                   ; (RPLACB A4 (CDDR A4)) ;
        CDR     A3,A2
        SCDR    A2,A4,[RETURN]          ; et retourne A1 ;
```

```
        ; Fonctions sur A-LISTES ;
        ;------------------------;

        ; (ASSQ at al) ;

        ENTRY   ASSQ,2SUBR

ASSQ:   FLIST   A2,,[JUMP (ASSQ2)]      ; la liste est vide ;
        MOVE    A1,A3                   ; prepare le retour ;
ASSQ1:  CAR     A2,A1                   ; A1 ← 1er élément ;
        CAR     A1,A4                   ; A4 ← la variable ;
        EQ      A4,A3,[RETURN]          ; c'est le bon ;
        CDR     A2,A2                   ; élément suivant ;
        TLIST   A2,,[JUMP (ASSQ1)]      ; la liste n'est pas vide ;
ASSQ2:  MOVE    NIL,A1,[RETURN]         ; retourne NIL ;

        ; (CASSQ at al) ;

        ENTRY   CASSQ,2SUBR

CASSQ1: CAR     A2,A3                   ; A1 ← 1er élément ;
        CAR     A3,A4                   ; A4 ← la variable ;
        EQ      A4,A1,[JUMP (CASSQ2)]   ; c'est le bon ;
        CDR     A2,A2                   ; élément suivant ;
CASSQ:  TLIST   A2,,[JUMP (CASSQ1)]     ; la liste n'est pas vide ;
        MOVE    NIL,A1,[RETURN]         ; retourne NIL ;
CASSQ2: CDR     A3,A1,[RETURN]          ; retourne le CDR de l'élément ;
```

```
            ; Fonctions numériques ;
            ;------------------------;
            ; calcul arithmétique ;
            ENTRY   ADD1,1SUBR
            ENTRY   SUB1,1SUBR
            ENTRY   ABS,1SUBR
            ENTRY   MINUS,1SUBR
            ENTRY   PLUS,2SUBR
            ENTRY   DIFFER,2SUBR
            ENTRY   TIMES,2SUBR
            ENTRY   QUO,2SUBR
            ENTRY   REM,2SUBR
ADD1:       ADD     '1,A1,[RETURN]
SUB1:       SUB     '1,A1,[RETURN]
ABS:        GE      A1,'0,[RETURN]
MINUS:      MOVE    '0,A2
            SUB     A1,A2
            MOVE    A2,A1,[RETURN]
PLUS:       ADD     A2,A1,[RETURN]
DIFFER:     SUB     A2,A1,[RETURN]
TIMES:      MUL     A2,A1,[RETURN]
QUO:        DIV     A2,A1,[RETURN]
REM:        REM     A2,A1,[RETURN]
            ; tests arithmétiques ;
            ENTRY   ZEROP,1SUBR
            ENTRY   NEROP,1SUBR
            ENTRY   MINUSP,1SUBR
            ENTRY   GT,2SUBR
            ENTRY   GE,2SUBR
            ENTRY   LT,2SUBR
            ENTRY   LE,2SUBR
ZEROP:      NEQ     A1,'0,[JUMP (FALSE)]
            NOP     ,,[RETURN]
NEROP:      EQ      A1,'0,[JUMP (FALSE)]
            NOP     ,,[RETURN]
MINUSP:     LT      A1,'0,[RETURN]
            NOP     ,,[JUMP (FALSE)]
GT:         GT      A1,A2,[RETURN]
            NOP     ,,[JUMP (FALSE)]
GE:         GE      A1,A2,[RETURN]
            NOP     ,,[JUMP (FALSE)]
LT:         LT      A1,A2,[RETURN]
            NOP     ,,[JUMP (FALSE)]
```

```
LE:     LE      A1,A2,[RETURN]
        NOP     ,,[JUMP (FALSE)]

        ; calculs logiques ;

        ENTRY   COMPL,1SUBR
        ENTRY   LOGAND,2SUBR
        ENTRY   LOGOR,2SUBR
        ENTRY   LOGXOR,2SUBR

COMPL:  LOGXOR '-1,A1,[RETURN]

LOGAND: LOGAND A2,A1,[RETURN]

LOGOR:  LOGOR  A2,A1,[RETURN]

LOGXOR: LOGXOR A2,A1,[RETURN]
```

```
        ; Fonctions d'entrée / sorties ;
        ;-------------------------------;

        ; Toutes les fonctions d'entrée/sortie de l'interprète ;
        ; V2I utilisent des fonctions toutes prêtes pour ;
        ; accélérer la simulation. ;
        ; Toutes ces fonctions sont masquées par les mêmes ;
        ; fonctions écrites en VCMC2 qui se trouvent sur les ;
        ; fichiers V2R (entrée) et V2P (sortie) ;

        ; Routines d'initialisation des E/S ;
        ; sur V2I elles sont inactives ;

READINI:NOP     ,,[RETURN]

PRININI:NOP     ,,[RETURN]

        ENTRY   READ,0SUBR

READ:   READ    A1,,[RETURN]

        ENTRY   READCH,0SUBR

READCH: READCH A1,,[RETURN]

        ;-----------------------;
        ; Fonctions de sortie ;
        ;-----------------------;

        ENTRY   PRIN,FSUBR,3
        ENTRY   PRINT,FSUBR,3
        ENTRY   PRINCH,1SUBR
        ENTRY   TERPRI,0SUBR

PRIN11: CDR     A1,TST,[CALL (EVCAR)]
        PRINI   '" "                    ; un espace entre chaque elements ;
        PRINI   A1                      ; imprime l'élément ;
        MOVE    A1,A2                   ; pour ramener en valeur ;
        POP     A1                      ; recupère le reste des elements ;
PRIN:   TLIST   A1,,[JUMP (PRIN11)]
        MOVE    A2,A1,[RETURN]

PRINT:  NOP     ,,[CALL (PRIN)]
        NOP     ,,[JUMP (OUTLIN)]

PRINCH: PRINI   A1,,[RETURN]

TERPRI: NOP     ,,[CALL (OUTLIN)]
        MOVE    NIL,A1,[RETURN]

        ; OUTLIN : vide la ligne en cours ;

OUTLIN: TERPRI ,,[RETURN]               ; et c'est tout ;

        ; PROBJ : fonction d'impression interne de A1 ;

PROBJ:  PRINI   A1,,[RETURN]

        ; PROBJT : fonction d'impression interne de A1 et TERPRI ;

PROBJT: PRINI   A1,,[JUMP (OUTLIN)]
```

```
                                            ; Fonctions système ;
                                            ;--------------------;

          ; termine l'interprète ;

          ENTRY   STOP,0SUBR

STOP:     PRINI   '"Bye",,[CALL (OUTLIN)]
          STOP

          ; PRSTACK ;
          ; Visualise la pile ;

          ENTRY   PRSTACK,1SUBR

PRSTACK:PRSTACKA1,,[RETURN]

          ;  CALL : appelle un sous-programme en LM ;
          ; (CALL adresse A1 A2 A3 A4) ;

          ENTRY   CALL,NSUBR

CALL:     CAR     A1,TST                    ; empile l'adresse ;
          CDR     A1,A4
          FLIST   A4,,[RETURN]              ; la liste est terminée ;
          CAR     A4,A1                     ; A1 ← le 1er argument ;
          CDR     A4,A4
          FLIST   A4,,[RETURN]              ; la liste est terminée ;
          CAR     A4,A2                     ; A2 ← le 2ème argument ;
          CDR     A4,A4
          FLIST   A4,,[RETURN]              ; la liste est terminée ;
          CAR     A4,A3                     ; A3 ← le 3ème argument ;
          CDR     A4,A4
          CAR     A4,A4,[RETURN]            ; A4 ← le 4ème argument ;
                                            ; et tombe sur l'adresse empilée ;

          ; (MEMORY adresse < valeur >) ;

          ENTRY   MEMORY,2SUBR

MEMORY:   SINDEX  A1                        ; registre index = adresse ;
          TLIST   A2,,[JUMP (MEMR1)]
          MOVEX   A2,'0                     ; force le mot mémoire ;
MEMR1:    XMOVE   '0,A1,[RETURN]            ; retourne la valeur du mot ;
```

```
          ; Données vives de l'interprète ;
          ;——————————————————————————————;


          ; dans l'interprète ;
                                            ; état de l'interprète : CHRONOLOGIE ;
EVALCH:   DATA    0                            ; indicateur de TRACE (NIL ou T) ;
EVALST:   DATA    NIL                          ; la forme à évaluer ;
FORME:    DATA    NIL                 ; pointeur sur le bloc de contrôle en pile ;
PBIND:    DATA    0                   ; sauvetage temporaire du pointeur de pile ;
SAVEP:    DATA    0                            ; sauvetage de CHRONOLOGIE ;
SAVECH:   DATA    0


          ))

; fin de l'interpréteur ;

'OK
```

```
; épilogue standard ;
; Déclaration du macro-caractère - (↑O) qui permet ;
; d'appeler l'interprète VCMC2I simulé sur VCMC2M  ;

(DE V2M (L)
   (VCMC2 [['MOVE [QUOTE L] 'A1] '(NOP NIL NIL CALL (SINGLE))])
   '(STOP))

(DMC "┘" () (ANACODE '(~interpreter) T T) ''ANACODE)

(DMC "—" ()
   [QUOTE
    (VCMC2
     [['MOVE [QUOTE (READ)] 'A1] '(NOP NIL NIL CALL (SINGLE)) '(STOP)])])

(DMC "î" ()
   [QUOTE
    (VCMC2
     [['MOVE [QUOTE (READ)] 'A1] '(NOP NIL NIL CALL (SINGLE)) '(STOP)] T)])

(DMC "Ô" ()
   [QUOTE
    (PROG2
       (SETQ ?instrace T ?contrace T ?prinstack T ?stepper T)
       (VCMC2
        [['MOVE [QUOTE (READ)] 'A1] '(NOP NIL NIL CALL (SINGLET)) '(STOP)])
       (SETQ ?instrace NIL ?contrace NIL ?prinstack NIL ?stepper NIL))])

(DMC "←" ()
   ; pour RECORD! ;
   [QUOTE
    (PROG2
       (SETQ ?instrace NIL ?stepper NIL)
       (VCMC2
        [['MOVE [QUOTE (READ)] 'A1] '(NOP NIL NIL CALL (SINGLET)) '(STOP)])
       (SETQ ?instrace NIL ?stepper NIL))])

(DMC "┐" () [QUOTE (VCMC2 ['(NOP NIL NIL JUMP (MAIN))] T T T)])

(DE RUNVCMC2 ()
   ; pour RECORD! ;
   (VCMC2 '((NOP NIL NIL JUMP (MAIN)) T T T)))

(DE ESCAPE.I ()
   ; une IT provoque un flip/flop des traces ;
   (SETQ
      ?instrace (NOT ?instrace)
      ?contrace (NOT ?contrace)
      ?prinstack (NOT ?prinstack)))

(PROGN
   (ANACODE '(~interpreter))
   (STATUS 1 1)
   (PRINT "...  Interprète VLISP - VCMC2 chargé.")
   (PRINT "     ┘  pour analyser l'interprète")
   (PRINT "     —s pour appeler EVAL avec s")
   (PRINT "     Ôs idem mais avec les traces V2M")
   (PRINT "     ←s trace simple de RECORD ")
   (PRINT "     îs idem mais avec les stats  V2I")
   (PRINT "     ┐  pour rentrer dans l'interprète"))
```

CROSS REFERENCE

Signification des codes associés aux numéros des lignes :

- \# définition de fonction de type DE DF DM DMC ou ENTRY
- & définition de fonction de type ESCAPE, ESCLOOP
- : définition d'étiquette dans PROG, DO, LAP ...
- , variable argument d'une fonction
- = variable affectée par SETQ ou SETQQ
- ' nom apparaissant dans une S-expression quotée
- \- code instruction assembleur

⌉	DMC	1965
$	DMC	49
ɑ	DMC	1948
←	DMC	1956
_	DMC	1938
↑	DMC	1943
⌋	DMC	1936
ABS	ENTRY	1728
ADD1	ENTRY	1726
ADDPROP	ENTRY	1646
AND	ENTRY	1095
APPEND	ENTRY	1439
APPEND1	ENTRY	1440
APPLY	ENTRY	549
APPLYN	ENTRY	550
ASSQ	ENTRY	1698
ATOM	ENTRY	1221
BOUNDP	ENTRY	1224
CAAAR	ENTRY	1289
CAADR	ENTRY	1290
CAAR	ENTRY	1285
CADAR	ENTRY	1291
CADDR	ENTRY	1292
CADR	ENTRY	1286
CALL	ENTRY	1883
CAR	ENTRY	1283
CASSQ	ENTRY	1711
CDAAR	ENTRY	1293
CDADR	ENTRY	1294
CDAR	ENTRY	1287
CDDAR	ENTRY	1295
CDDDR	ENTRY	1296
CDDR	ENTRY	1288
CDR	ENTRY	1284
CHRONOLOGY	ENTRY	782
CNTH	ENTRY	1372
CODE	DF	44
COMPL	ENTRY	1788
COND	ENTRY	1078
CONS	ENTRY	1398
COPY	ENTRY	1473
CVAL	ENTRY	1021
DE	ENTRY	980
DELQ	ENTRY	1460
DF	ENTRY	981
DIFFER	ENTRY	1731

```
DM              ENTRY 982
EPROGN          ENTRY 667
EQ              ENTRY 1245
EQUAL           ENTRY 1256
ERROR           ENTRY 80
ESCAPE          ENTRY 884
ESCAPE.I        DE    1971
EVAL            ENTRY 160
EVLIS           ENTRY 691
EVLIS*          ENTRY 715
EXIT            ENTRY 761
EXITCHRONOLOG   ENTRY 791
FINDCHRONOLOG   ENTRY 808
FREVERSE        ENTRY 1572
FREVERSE*       ENTRY 1585
FTYPE           ENTRY 1024
FVAL            ENTRY 1023
GE              ENTRY 1761
GET             ENTRY 1645
GT              ENTRY 1760
IF              ENTRY 1060
IFN             ENTRY 1061
INTERNAL        ENTRY 657
KWOTE           ENTRY 1421
LAMBDA          ENTRY 656
LAST            ENTRY 1359
LE              ENTRY 1763
LENGTH          ENTRY 1383
LESCAPE         ENTRY 762
LET             ENTRY 832
LETF            ENTRY 937
LIST            ENTRY 690
LIST*           ENTRY 714
LISTP           ENTRY 1223
LOGAND          ENTRY 1789
LOGOR           ENTRY 1790
LOGXOR          ENTRY 1791
LT              ENTRY 1762
MAP             ENTRY 1164
MAPC            ENTRY 1163
MCONS           ENTRY 1410
MEMBER          ENTRY 1342
MEMORY          ENTRY 1901
MEMQ            ENTRY 1332
MINUS           ENTRY 1729
MINUSP          ENTRY 1759
NCONC           ENTRY 1625
NCONC1          ENTRY 1626
NCONS           ENTRY 1400
NEQ             ENTRY 1246
NEQUAL          ENTRY 1257
NEROP           ENTRY 1758
NEWL            ENTRY 1613
NEXTL           ENTRY 1603
NOT             ENTRY 1220
NTH             ENTRY 1371
NULL            ENTRY 1219
NUMBP           ENTRY 1222
OR              ENTRY 1094
PLIST           ENTRY 1022
PLUS            ENTRY 1730
PRIN            ENTRY 1831
PRINCH          ENTRY 1833
```

F-47

PRINT	ENTRY	[illegible]32
PROG1	ENTRY	6[illegible]
PROGN	ENTRY	666
PRSTACK	ENTRY	1876
PUT	ENTRY	1647
QUO	ENTRY	1733
QUOTE	ENTRY	1282
QUOTE	ENTRY	648
RDE	ENTRY	991
RDF	ENTRY	992
RDM	ENTRY	993
READ	ENTRY	1819
READCH	ENTRY	1823
REM	ENTRY	1734
REMPROP	ENTRY	1648
REVERSE	ENTRY	1428
REVERT	ENTRY	1002
RPLACA	ENTRY	1527
RPLACB	ENTRY	1529
RPLACD	ENTRY	1528
RUNVCMC2	DE	1967
SELECTQ	ENTRY	1135
SELF	ENTRY	729
SET	ENTRY	1555
SETQ	ENTRY	1540
SETQQ	ENTRY	1563
STEPEVAL	ENTRY	216
STOP	ENTRY	1868
SUB1	ENTRY	1727
SUBST	ENTRY	1484
SUBST*	ENTRY	1499
SYNONYM	ENTRY	1044
TERPRI	ENTRY	1834
TIMES	ENTRY	1732
TOPLEVEL	ENTRY	136
UNTIL	ENTRY	1117
V2M	DE	1932
WHERE	ENTRY	883
WHILE	ENTRY	1116
XCONS	ENTRY	1399
ZEROP	ENTRY	1757

1	*EOS*	102'	122'									
2	*MARK*	74'	194'	326'	375'	400'	465'	510'	539'	578'	624'	745'
		752'	774'	802'	820'	1167'	1184'	1192'	1208'			
3	0SUBR	136	1819	1823	1834	1868						
4	1SUBR	216	667	691	715	782	1002	1021	1219	1220	1221	1222
		1223	1224	1283	1284	1285	1286	1287	1288	1289	1290	1291
		1292	1293	1294	1295	1296	1359	1383	1400	1421	1473	1726
		1727	1728	1729	1757	1758	1759	1788	1833	1876		
5	2SUBR	549	1022	1023	1024	1044	1245	1246	1256	1257	1332	1342
		1371	1372	1398	1399	1428	1439	1440	1460	1527	1528	1529
		1555	1572	1585	1625	1626	1645	1648	1698	1711	1730	1731
		1732	1733	1734	1760	1761	1762	1763	1789	1790	1791	1901
6	3SUBR	160	1484	1499	1646	1647						
7	?contrace	1951=	1954=	1975=	1975							
8	?instrace	1951=	1954=	1960=	1963=	1974=	1974					
9	?prinstack	1951=	1954=	1976=	1976							
10	?stepper	1951=	1954=	1960=	1963=							
11	@	76	104	105	108	109	123	126	127	128	163	164
		165	167	168	169	170	172	181	183	205	210	223

	237	238	239	241	242	401	437	466	472	480	492
	493	494	500	508	509	518	527	528	622	623	639
	661	732	733	742	765	765	766	767	771	771	786
	787	795	796	797	799	814	815	817	866	911	913
	952	954	1171	1188	1196	1212	1260	1272			
12 A1	71	83	84	86	87	87	88	89	92	94	106
	111	113	114	129	147	179	179	183	184	191	191
	192	206	208	219	220	221	222	226	227	228	229
	249	250	250	288	288	289	291	291	300	301	302
	303	336	337	338	339	343	344	345	346	347	349
	380	383	389	390	395	396	397	398	401	402	418
	430	430	435	441	441	484	486	487	500	513	515
	519	552	553	553	554	555	556	557	558	585	586
	591	594	595	598	598	599	600	600	601	605	605
	607	608	608	611	612	612	629	630	630	638	640
	650	650	661	670	672	675	682	683	684	695	696
	698	699	701	703	705	707	708	718	719	720	721
	754	776	785	786	787	804	811	812	822	834	835
	835	842	843	847	847	848	851	852	853	854	855
	857	858	859	860	861	866	867	868	886	887	890
	891	894	895	897	899	919	920	920	939	941	944
	945	948	951	962	963	963	964	966	970	972	973
	973	974	975	1004	1007	1008	1010	1026	1026	1028	1029
	1031	1032	1032	1035	1036	1036	1039	1040	1040	1047	1049
	1051	1063	1066	1067	1069	1072	1073	1080	1083	1084	1087
	1097	1100	1101	1103	1104	1107	1108	1119	1120	1121	1121
	1122	1123	1126	1127	1128	1128	1129	1130	1137	1139	1147
	1153	1166	1168	1169	1169	1170	1178	1179	1186	1191	1193
	1194	1194	1195	1210	1227	1228	1230	1231	1233	1234	1236
	1237	1239	1241	1248	1249	1251	1252	1262	1264	1265	1265
	1269	1270	1271	1273	1275	1298	1298	1299	1299	1300	1300
	1302	1302	1303	1303	1304	1304	1306	1306	1307	1307	1308
	1308	1310	1310	1311	1311	1312	1312	1314	1314	1315	1315
	1316	1316	1318	1318	1319	1319	1320	1320	1322	1322	1323
	1323	1324	1324	1326	1326	1327	1327	1328	1328	1335	1338
	1345	1347	1348	1351	1352	1353	1361	1362	1364	1365	1366
	1377	1378	1379	1386	1386	1388	1389	1402	1404	1406	1412
	1413	1416	1417	1423	1424	1430	1431	1431	1433	1434	1447
	1451	1451	1452	1455	1462	1462	1463	1464	1467	1467	1468
	1475	1476	1477	1477	1478	1479	1488	1489	1493	1494	1501
	1502	1503	1504	1506	1511	1512	1513	1514	1515	1516	1517
	1517	1518	1519	1531	1532	1535	1536	1542	1543	1544	1544
	1545	1548	1557	1558	1565	1566	1566	1567	1567	1568	1575
	1576	1577	1578	1580	1581	1588	1589	1592	1594	1597	1598
	1599	1605	1607	1615	1616	1616	1619	1620	1630	1631	1632
	1633	1634	1636	1637	1654	1656	1657	1669	1671	1672	1673
	1676	1701	1702	1703	1707	1715	1718	1719	1736	1738	1740
	1742	1743	1745	1747	1749	1751	1753	1765	1768	1771	1774
	1777	1780	1783	1793	1795	1797	1799	1821	1825	1836	1838
	1839	1840	1841	1842	1847	1850	1858	1862	1878	1885	1886
	1888	1903	1906	1933'	1941'	1946'	1953'	1962'			
13 A2	72	75	76	130	162	165	206	207	223	224	249
	252	258	259	284	285	286	290	296	301	302	306
	309	309	310	311	311	312	322	325	328	338	348
	374	381	382	384	385	385	386	387	388	395	397
	399	400	406	423	424	425	426	483	485	488	507
	511	516	516	517	524	525	534	541	552	558	589
	595	607	609	609	611	613	614	614	621	625	640
	672	673	674	706	707	731	743	751	795	799	813
	817	836	837	838	846	854	855	863	864	865	867
	886	888	889	889	890	895	898	906	907	908	909
	910	919	921	939	940	942	943	943	946	949	964
	965	967	972	975	1005	1030	1031	1034	1035	1038	1039

		1046	1048	1050	1064	1065	1067	1070	1071	1073	1080	1081
		1082	1083	1086	1097	1098	1099	1104	1105	1106	1110	1124
		1131	1140	1141	1143	1148	1150	1152	1172	1176	1179	1197
		1201	1203	1239	1240	1248	1251	1263	1266	1267	1267	1268
		1271	1334	1336	1336	1337	1338	1344	1346	1346	1349	1350
		1351	1362	1363	1364	1366	1375	1375	1376	1379	1385	1387
		1389	1402	1404	1413	1414	1415	1432	1434	1442	1453	1454
		1465	1487	1501	1507	1510	1531	1532	1533	1534	1542	1546
		1549	1557	1558	1565	1568	1576	1577	1581	1589	1591	1593
		1594	1595	1598	1605	1606	1609	1617	1618	1620	1628	1631
		1636	1658	1659	1666	1674	1678	1679	1682	1684	1688	1689
		1690	1691	1700	1702	1705	1705	1706	1713	1716	1716	1717
		1741	1742	1743	1745	1747	1749	1751	1753	1774	1777	1780
		1783	1795	1797	1799	1839	1842	1891	1904	1905		
14	A3	73	74	75	166	169	225	259	261	296	297	298
		310	321	322	323	324	325	347	407	429	434	449
		467	476	482	485	495	511	512	513	514	529	530
		531	532	557	560	577	586	587	588	599	613	615
		615	625	626	628	629	631	633	634	635	636	637
		638	670	697	700	702	703	704	732	742	749	768
		769	770	798	816	839	840	841	850	859	860	862
		868	897	910	921	947	950	966	967	968	974	984
		985	986	995	996	997	1006	1009	1012	1046	1047	1048
		1049	1050	1051	1065	1071	1085	1088	1138	1140	1141	1142
		1142	1144	1145	1145	1149	1151	1173	1177	1187	1198	1202
		1211	1334	1335	1430	1432	1444	1445	1446	1449	1450	1454
		1464	1465	1466	1486	1487	1488	1490	1491	1491	1492	1502
		1504	1505	1508	1509	1513	1533	1535	1547	1548	1575	1578
		1588	1590	1591	1592	1595	1599	1606	1607	1608	1608	1609
		1618	1619	1633	1634	1635	1659	1666	1675	1684	1686	1687
		1687	1688	1690	1701	1704	1713	1714	1719	1894		
15	A4	323	324	373	374	394	433	460	461	462	463	464
		465	473	474	475	481	486	501	504	507	519	534
		554	580	626	627	631	632	633	733	734	735	741
		744	747	748	843	844	845	846	848	849	850	852
		856	856	857	893	896	909	918	1008	1009	1010	1011
		1011	1012	1139	1144	1146	1150	1175	1176	1177	1178	1180
		1180	1181	1182	1183	1184	1185	1200	1201	1202	1203	1204
		1204	1205	1206	1207	1208	1209	1447	1448	1449	1450	1534
		1536	1655	1657	1658	1660	1660	1661	1661	1662	1667	1668
		1668	1669	1673	1674	1675	1676	1680	1681	1681	1682	1685
		1686	1689	1691	1703	1704	1714	1715	1886	1887	1888	1889
		1889	1890	1891	1892	1892	1893	1894	1895	1895	1896	1896
16	ABS	1728#	1740:									
17	ADD	242-	1387-	1736-	1745-							
18	ADD1	1726#	1736:									
19	ADDPROP	968	1646#	1671:	1680							
20	ANACODE	1936	1936'	1979								
21	AND	1095#	1103:									
22	AND1	1104:	1108									
23	APPEN2	1447:	1452									
24	APPEN3	1446	1452:									
25	APPEN4	1453	1455:									
26	APPEND	1439#	1444:									
27	APPEND1	1440#	1442:									
28	APPEX1	626:	634									
29	APPEX2	625	634:									
30	APPEX3	635	639:									
31	APPEXP	572	595	621:								
32	APPLIN	559:	601									
33	APPLY	243	486	549#	554:	591	1188	1212				
34	APPLY1	567	605:									
35	APPLY2	568	607:									

36	APPLY3	569	611:									
37	APPLYI	588	597:									
38	APPLYL	587	593:									
39	APPLYN	550#	552:									
40	APPLYS	555	585:									
41	ASSQ	1698#	1700:									
42	ASSQ1	1702:	1706									
43	ASSQ2	1700	1707:									
44	ATOM	1221#	1230:									
45	BOUNDP	1224#	1239:									
46	CAAAR	1289#	1298:									
47	CAADR	1290#	1302:									
48	CAAR	1285#	1299:									
49	CADAR	1291#	1306:									
50	CADDR	1292#	1310:									
51	CADR	1286#	1307:									
52	CALL	64	84	87	90	92	106	107	111	113	116	124
		125	130	140	141	144	145	146	149	208	220	221
		225	227	228	284	301	336	337	343	345	346	381
		388	441	486	524	589	674	682	683	696	703	719
		720	794	811	812	892	895	944	950	968	1005	1063
		1069	1084	1099	1106	1121	1122	1128	1129	1137	1150	1188
		1212	1260	1267	1346	1415	1416	1467	1477	1478	1491	1493
		1508	1517	1518	1545	1616	1666	1678	1684	1836	1844	1849
		1870	1883#	1885:	1933'	1941'	1946'	1953'	1962'			
53	CAR	83-	87-	179-	249-	286	291-	296-	310-	312-	374-	384-
		507-	511-	553-	586-	599-	601-	605-	608-	609-	612-	614-
		615-	625-	626-	629-	650-	703-	835-	843-	848-	852-	857-
		886-	888-	890-	919-	939-	942-	943-	963-	973-	1008-	1011-
		1067-	1073-	1083-	1142-	1144-	1166-	1168-	1178-	1191-	1193-	1265-
		1267-	1283#	1298-	1299-	1300:	1300-	1303-	1304-	1306-	1308-	1312-
		1314-	1315-	1319-	1322-	1334-	1346-	1374	1430-	1447-	1464-	1477-
		1491-	1517-	1533-	1543-	1565-	1567-	1605-	1607-	1615-	1658-	1669-
		1688-	1702-	1703-	1713-	1714-	1885-	1888-	1891-	1894-	1896-	
54	CASSQ	1711#	1717:									
55	CASSQ1	1713:	1717									
56	CASSQ2	1715	1719:									
57	CDAAR	1293#	1314:									
58	CDADR	1294#	1318:									
59	CDAR	1287#	1315:									
60	CDDAR	1295#	1322:									
61	CDDDR	1296#	1326:									
62	CDDR	1288#	1327:									
63	CDR	84-	86-	250-	288-	306-	309-	311-	336-	343-	345-	380-
		385-	430-	441-	516-	552-	594-	598-	600-	607-	611-	613-
		630-	633-	672-	682-	696-	702-	719-	811-	834-	847-	851-
		856-	862-	863-	887-	889-	895-	920-	940-	941-	972-	1010-
		1064-	1065-	1070-	1071-	1082-	1084-	1097-	1104-	1121-	1128-	1137-
		1141-	1145-	1169-	1180-	1194-	1204-	1264-	1266-	1284#	1302-	1307-
		1310-	1311-	1316-	1318-	1320-	1323-	1324-	1326-	1327-	1328:	1328-
		1336-	1349-	1362-	1364-	1375-	1386-	1413-	1431-	1451-	1455-	1462-
		1467-	1476-	1490-	1516-	1534-	1544-	1545-	1566-	1575-	1588-	1591-
		1594-	1608-	1616-	1634-	1660-	1661-	1668-	1681-	1686-	1687-	1690-
		1705-	1716-	1719-	1836-	1886-	1889-	1892-	1895-			
64	CHRON1	785	787:									
65	CHRONOLOGY	782#	784:									
66	CNTH	1372#	1374:									
67	CODE	44#	54									
68	COMPL	1788#	1793:									
69	COND	1078#	1080:									
70	COND1	1081:	1087									
71	CONS	75-	76-	721-	865-	967-	1176-	1179-	1201-	1203-	1398#	1402:
		1404-	1417-	1424-	1432-	1468-	1479-	1494-	1519-	1674-	1675-	

72	COPY	1473	1475:	1477	1478	1502						
73	CVAL	191-	322-	323-	395-	512-	627-	636-	1021#	1026:	1026-	1239-
		1606-	1618-									
74	DATA	185-	186-	187-	254-	255-	256-	270-	271-	272-	273-	274-
		275-	276-	277-	278-	279-	280-	412-	413-	414-	415-	453-
		454-	455-	456-	565-	566-	567-	568-	569-	570-	571-	572-
		573-	574-	575-	737-	738-	739-	740-	1914-	1915-	1916-	1917-
		1918-	1919-									
75	DE	980#	984:	1932	1967	1971						
76	DEF	972:	984	985	986							
77	DELQ	1460#	1463:	1467								
78	DELQ3	1462:	1465									
79	DF	44	981#	985:								
80	DIFFER	1731#	1747:									
81	DISPT	184-	252-									
82	DIV	1751-										
83	DM	982#	986:									
84	DMC	49	1936	1938	1943	1948	1956	1965				
85	ENTRY	80	136	160	216	549	550	648	656	657	666	667
		680	690	691	714	715	729	761	762	782	791	808
		832	883	884	937	980	981	982	991	992	993	1002
		1021	1022	1023	1024	1044	1060	1061	1078	1094	1095	1116
		1117	1135	1163	1164	1219	1220	1221	1222	1223	1224	1245
		1246	1256	1257	1282	1283	1284	1285	1286	1287	1288	1289
		1290	1291	1292	1293	1294	1295	1296	1332	1342	1359	1371
		1372	1383	1398	1399	1400	1410	1421	1428	1439	1440	1460
		1473	1484	1499	1527	1528	1529	1540	1555	1563	1572	1585
		1603	1613	1625	1626	1645	1646	1647	1648	1698	1711	1726
		1727	1728	1729	1730	1731	1732	1733	1734	1757	1758	1759
		1760	1761	1762	1763	1788	1789	1790	1791	1819	1823	1831
		1832	1833	1834	1868	1876	1883	1901				
86	EPROGN	667#	671:									
87	EQ	74-	297-	298-	324-	418-	527-	530-	587-	588-	734-	766-
		769-	796-	799-	814-	1139-	1245#	1248:	1248-	1271-	1335-	1465-
		1487-	1659-	1704-	1715-	1768-						
88	EQUAL	1256#	1260:	1346	1508							
89	EQUAL1	1263:	1270									
90	EQUAL2	1260	1267	1270:								
91	ERESC	527	538:									
92	ERRO2	86:	89									
93	ERRO4	85	88:									
94	ERROR	71'	80#	82:								
95	ESCAPE	884#	918:									
96	ESCAPE.I	1971#										
97	EVAL	160#	162:	225	892							
98	EVAL0	271	332:									
99	EVAL1	177:	272									
100	EVAL2	273	335:									
101	EVAL3	274	342:									
102	EVALA1	113	146	180:	301	502	589	703	868	1067	1073	
103	EVALAN	166	173	182:								
104	EVALAT	185	191:									
105	EVALCH	105	128	164	165	168	223	238	242	493	732	742
		765	771	786	787	795	799	817	1914:			
106	EVALF	276	351:									
107	EVALFAT	254	258:									
108	EVALFLI	256	295:									
109	EVALFNB	255	284:									
110	EVALFU	251:	303	325	944	950						
111	EVALI	298	308:									
112	EVALIN	260:	312									
113	EVALIS	187	248:									
114	EVALL	297	305:									

115	EVALL2	162	166:									
116	EVALL3	165	169:									
117	EVALN	275	693:									
118	EVALN1	288:	290									
119	EVALN2	287	290:									
120	EVALNB	186	199:									
121	EVALST	104	123	163	167	169	181	205	210	237	494	1915:
122	EVALT	181	204:									
123	EVCAR	178:	284	336	337	343	345	346	381	673	674	682
		696	719	811	895	1063	1069	1084	1098	1099	1105	1106
		1122	1129	1137	1414	1415	1545	1616	1836			
124	EVESC	280	523:	575								
125	EVESC1	526:	531	534								
126	EVESC3	530	532:									
127	EVESC4	532	534:									
128	EVEXP	277	306	373:	744							
129	EVEXP1	380:	386									
130	EVEXP2	375	386:									
131	EVEXP3	387	394:									
132	EVEXP4	395:	400									
133	EVEXP5	394	399:									
134	EVEXP6	408:	420	426								
135	EVEXP9	418	428:									
136	EVEXPF	435:	519	640								
137	EVEXPG	441:										
138	EVEXPN	408	414	415	432:							
139	EVFEX2	511:	517									
140	EVFEX3	510	517:									
141	EVFEXB	502	505:									
142	EVFEXP	278	504:	573								
143	EVLIS	388	691#	695:								
144	EVLIS*	715#	717:									
145	EVLIS1	700:	707									
146	EVLIS2	700	708:									
147	EVMAC	279	500:	574								
148	EXCHER	796	802:									
149	EXICH1	796:	799									
150	EXICH2	798	799:									
151	EXIT	761#	765:									
152	EXIT1	766:	770									
153	EXIT2	770:	771									
154	EXIT3	769	771:									
155	EXITCHRONOLOG	94	791#	793:								
156	EXITER	766	774:									
157	FALSE	1028	1187	1211	1273:	1376	1667	1685	1765	1768	1772	1775
		1778	1781	1784								
158	FATOM	555-	1028-	1486-								
159	FICHER	814	820:									
160	FINCH1	814:	817									
161	FINCH2	816	817:									
162	FINDCHRONOLOG	808#	810:									
163	FLIST	289-	673-	695-	700-	718-	1081-	1098-	1103-	1105-	1143-	1146-
		1230-	1263-	1361-	1363-	1376-	1412-	1414-	1453-	1475-	1515-	1590-
		1593-	1667-	1680-	1685-	1700-	1887-	1890-	1893-			
164	FNIL	181-	785-	1072-	1088-	1100-	1123-	1347-				
165	FORME	76	109	127	183	500	661	866	1916:			
166	FREV1	1575:	1580	1597								
167	FREV1*	1588:										
168	FREV9*	1593	1599:									
169	FREVERSE	1572#	1579:									
170	FREVERSE*	1585#	1596:									
171	FSUBR	648	656	657	666	680	690	714	729	761	762	791
		808	832	883	884	937	980	981	982	991	992	993

F. 53

		1060	1061	1078	1094	1095	1116	1117	1135	1282	1410	1540
		1563	1603	1613	1831	1832						
172	FTYP	259-	557-	906-	966-	1040-	1048-					
173	FTYP1	1038	1040:									
174	FTYPE	1024#	1038:									
175	FVAL	258-	556-	907-	964-	1023#	1034:	1036-	1046-			
176	FVAL1	1034	1036:									
177	GE	1740-	1761#	1777:	1777-							
178	GET	1005	1645#	1665:								
179	GETI	1654:	1666	1678	1684							
180	GETI0	1655:	1656									
181	GETI1	1654	1656:									
182	GETI2	1658:	1662									
183	GETI3	1657	1662:									
184	GT	1378-	1760#	1774:	1774-							
185	IF	1060#	1063:									
186	IFN	1061#	1069:									
187	INTERNAL	298'	588'	657#	660:	968'	1005'					
188	IT	114'	147'									
189	ITSOFT	76	130	208	236:							
190	JUMP	74	75	76	85	89	94	132	162	165	166	173
		181	195	243	286	287	290	297	298	303	306	312
		324	325	328	375	386	387	394	400	408	418	420
		426	430	462	465	502	510	517	519	527	530	531
		532	534	541	555	580	585	587	588	591	595	601
		625	634	635	640	673	675	700	707	734	742	744
		745	747	749	754	766	769	770	771	772	776	785
		796	798	799	804	814	816	817	822	842	844	850
		861	868	891	893	914	921	955	984	985	986	995
		996	997	1012	1028	1030	1034	1038	1066	1067	1072	1073
		1087	1088	1098	1100	1101	1103	1105	1107	1108	1119	1123
		1126	1130	1138	1139	1143	1146	1154	1155	1167	1170	1173
		1175	1176	1184	1187	1189	1192	1195	1198	1200	1201	1208
		1211	1213	1227	1230	1236	1240	1248	1251	1263	1270	1335
		1337	1347	1350	1365	1374	1376	1378	1385	1388	1414	1433
		1446	1452	1453	1465	1486	1501	1502	1511	1549	1580	1593
		1597	1630	1632	1635	1654	1656	1657	1662	1667	1680	1685
		1700	1706	1715	1717	1765	1768	1772	1775	1778	1781	1784
		1841	1845	1862	1904	1965'	1969'					
191	JUMPX	261-	410-	451-	560-	736-						
192	KWOTE	1421#	1423:									
193	L	44,	44	1932,	1933							
194	LAMBDA	297'	587'	656#	659:	865'						
195	LAST	1359#	1361:									
196	LAST1	1362:	1365									
197	LE	286-	1763#	1783:	1783-							
198	LENGT1	1386:	1388									
199	LENGT2	1385	1388:									
200	LENGTH	1383#	1385:									
201	LESCAPE	762#	764:									
202	LET	832#	834:									
203	LET2	842:	861									
204	LET4	844	851:									
205	LET8	842	850	862:								
206	LETF	937#	939:									
207	LIST	690#	694:									
208	LIST*	714#	718:	720								
209	LISTP	1223#	1236:									
210	LOGAND	1789#	1795:	1795-								
211	LOGOR	1790#	1797:	1797-								
212	LOGXOR	1791#	1793-	1799:	1799-							
213	LT	1762#	1771-	1780:	1780-							
214	MAIN	122:	1965'	1969'								

```
215 MAIN1      128:  132
216 MAP        1164# 1191:
217 MAP1       1193: 1195
218 MAP11      1192  1194:
219 MAP2       1196: 1213
220 MAP3       1200: 1208
221 MAP4       1200  1202:
222 MAP5       1201  1205:
223 MAP6       1198  1207:
224 MAPC       1163# 1166:
225 MAPC1      1168: 1170
226 MAPC11     1167  1169:
227 MAPC2      1171: 1189
228 MAPC3      1175: 1184
229 MAPC4      1175  1177:
230 MAPC5      1176  1181:
231 MAPC6      1173  1183:
232 MCONS      1410# 1412:
233 MCONS1     1413: 1416
234 MEMB1      1344: 1350
235 MEMB3      1347  1352:
236 MEMBER     1150  1342# 1350:
237 MEMORY     1901# 1903:
238 MEMQ       1332# 1337:
239 MEMQ1      1334: 1337
240 MEMQ2      1335  1338:
241 MEMR1      1904  1906:
242 MERROR     63:
243 MINUS      1729# 1741:
244 MINUSP     1759# 1771:
245 MOVE       71-   72-   94-   104-  105-  108-  109-  123-  126-  127-  128-
               129-  130-  165-  169-  183-  205-  206-  208-  210-  220-  223-
               225-  227-  301-  302-  325-  338-  347-  401-  449-  485-  486-
               500-  515-  519-  531-  532-  554-  558-  595-  640-  661-  670-
               732-  733-  765-  770-  786-  787-  795-  798-  816-  836-  839-
               855-  860-  866-  893-  897-  918-  921-  944-  968-  984-  985-
               986-  995-  996-  997-  1005- 1080- 1140- 1150- 1172- 1173- 1177-
               1197- 1198- 1202- 1228- 1231- 1234- 1237- 1241- 1249- 1252- 1262-
               1273- 1275- 1338- 1351- 1366- 1379- 1385- 1389- 1434- 1444- 1450-
               1488- 1502- 1504- 1513- 1542- 1558- 1577- 1578- 1581- 1598- 1599-
               1631- 1633- 1655- 1666- 1684- 1701- 1707- 1718- 1741- 1743- 1839-
               1842- 1850- 1933' 1941' 1946' 1953' 1962'
246 MOVEX      1905-
247 MUL        1749-
248 NAN        1263  1272:
249 NCONC      1625# 1630:
250 NCONC1     1626# 1628:
251 NCONC2     1630  1632:
252 NCONC3     1633: 1635
253 NCONC4     1632  1634:
254 NCONS      1400# 1406:
255 NEQ        192-  400-  408-  465-  534-  742-  745-  771-  817-  1184- 1208-
               1240- 1246# 1251: 1251- 1501- 1765-
256 NEQUAL     1257# 1259:
257 NEROP      1758# 1768:
258 NEWL       1613# 1615:
259 NEXTL      1603# 1605:
260 NOP        49'   64-   90-   107-  118-  124-  125-  132-  140-  144-  145-
               146-  151-  199-  287-  330-  332-  351-  535-  794-  800-  818-
               892-  1089- 1155- 1189- 1213- 1550- 1663- 1766- 1769- 1772- 1775-
               1778- 1781- 1784- 1815- 1817- 1844- 1845- 1849- 1933' 1941' 1946'
               1953' 1962' 1965' 1969'
261 NOT        1220# 1227: 1259  1974  1975  1976
```

262 NSUBR	80	550	1163	1164	1883						
263 NTH	137[illegible]	1374	1377:								
264 NTH1	1375:	1378									
265 NULL	1219#	1226:									
266 NUMBP	1222#	1233:									
267 OK	1926'										
268 OR	1094#	1097:	1101								
269 OUTLIN	64	84	90	140	141	1845	1849	1854:	1862	1870	
270 PBIND	108	126	170	172	239	241	401	437	466	472	480
	492	509	527	528	623	733	766	767	796	797	814
	815	911	913	952	954	1917:					
271 PLIST	1022#	1028:	1032-	1657-	1673-						
272 PLIST1	1030	1032:									
273 PLUS	1730#	1745:									
274 POP	73-	88-	92-	113-	222-	229-	285-	303-	321-	339-	348-
	349-	382-	398-	399-	406-	423-	424-	425-	426-	460-	461-
	462-	464-	466-	472-	473-	480-	481-	487-	488-	492-	493-
	494-	501-	504-	525-	577-	591-	621-	675-	684-	697-	704-
	706-	708-	741-	747-	748-	749-	813-	861-	896-	898-	899-
	946-	947-	951-	970-	1006-	1085-	1086-	1101-	1108-	1110-	1124-
	1131-	1138-	1151-	1152-	1182-	1185-	1186-	1206-	1209-	1210-	1268-
	1269-	1348-	1352-	1353-	1492-	1505-	1509-	1510-	1512-	1514-	1546-
	1547-	1617-	1637-	1679-	1840-						
275 POPJ	330:	449	566	570	571						
276 PRET	1100	1107	1110:								
277 PRIN	1831#	1841:	1844								
278 PRIN11	1836:	1841									
279 PRINCH	1833#	1847:									
280 PRINI	63-	85-	91-	110-	116-	141-	142-	149-	1837-	1838-	1847-
	1858-	1862-	1870-								
281 PRININI	107	125	1817:								
282 PRINT	1832#	1844:	1981	1982	1983	1984	1985	1986	1987		
283 PROBJ	87	220	227	1858:							
284 PROBJT	92	111	116	145	149	221	228	1862:			
285 PROG1	680#	682:									
286 PROG2	1950	1959									
287 PROGN	430	441	524	666#	672:	675	683	772	794	812	914
	955	1121	1128	1978							
288 PROGNA3	670:	1066	1072	1088	1139	1143	1155				
289 PRSTACK	65-	93-	1876#	1878:	1878-						
290 PRSTAT	66-	131-									
291 PTYP	1050-										
292 PUSH	106-	163-	164-	167-	168-	170-	171-	173-	194-	195-	219-
	226-	237-	238-	239-	240-	243-	284-	300-	326-	327-	328-
	375-	381-	388-	390-	434-	435-	436-	467-	476-	483-	484-
	495-	502-	509-	510-	514-	518-	524-	539-	540-	541-	578-
	579-	580-	589-	623-	624-	628-	637-	639-	674-	699-	701-
	744-	752-	753-	754-	772-	774-	775-	776-	802-	803-	804-
	820-	821-	822-	838-	841-	894-	908-	911-	912-	914-	949-
	950-	952-	953-	955-	962-	1004-	1063-	1069-	1099-	1106-	1119-
	1126-	1147-	1148-	1149-	1167-	1192-	1259-	1344-	1345-	1374-	1415-
	1446-	1466-	1489-	1503-	1506-	1507-	1508-	1632-	1678-		
293 PUT	1647#	1678:									
294 QUO	1733#	1751:									
295 QUOTE	648#	650:	1282#	1424'	1933	1939	1941	1944	1946	1949	1953
	1958	1962	1965								
296 RDE	991#	995:									
297 RDEF	962:	995	996	997							
298 RDF	992#										
299 RDM	993#										
300 READ	144	1819#	1821:	1821-	1941	1946	1953	1962			
301 READCH	1823#	1825:	1825-								
302 READINI	106	124	1815:								

303	REM	1734# 1753: 1753-
304	REMPROP	1012 1648# 1684:
305	RETURN	118 151 192 199 210 229 289 291 330 332 339 349 351 467 476 488 495 535 605 609 615 650 661 684 695 708 718 721 786 787 800 818 975 1007 1026 1029 1032 1036 1040 1051 1081 1089 1110 1124 1131 1228 1231 1233 1234 1237 1241 1249 1252 1262 1271 1273 1275 1300 1304 1308 1312 1316 1320 1324 1328 1338 1351 1353 1361 1363 1366 1379 1389 1402 1404 1406 1412 1417 1424 1434 1455 1463 1468 1475 1479 1487 1488 1404 1513 1515 1519 1531 1532 1536 1550 1558 1568 1581 1590 1598 1599 1609 1620 1631 1637 1655 1659 1663 1669 1671 1672 1676 1682 1691 1704 1707 1718 1719 1736 1738 1740 1743 1745 1747 1749 1751 1753 1766 1769 1771 1774 1777 1780 1783 1793 1795 1797 1799 1815 1817 1821 1825 1842 1847 1850 1854 1858 1878 1887 1890 1893 1896 1906
306	REV1	1430: 1433
307	REVERSE	1428# 1433:
308	REVERT	1002# 1004:
309	REVIVE	112- 117- 143- 150-
310	RPLACA	53 1527# 1531:
311	RPLACB	1529# 1533:
312	RPLACD	1528# 1532:
313	RUNVCMC2	1967#
314	SAVECH	765 771 1919:
315	SAVEP	508 518 622 639 1171 1188 1196 1212 1260 1272 1918:
316	SCAR	867- 1531- 1535- 1682- 1689-
317	SCDR	707- 846- 850- 854- 859- 868- 1449- 1454- 1532- 1536- 1576- 1589- 1592- 1595- 1636- 1691-
318	SCVAL	114- 147- 397- 463- 513- 631- 638- 1548- 1557- 1568- 1609- 1620-
319	SELEC2	1139: 1146
320	SELEC3	1140: 1154
321	SELEC4	1138 1141:
322	SELECTQ	1135# 1137:
323	SELF	729# 731:
324	SELF1	734: 747 749
325	SELFER	734 751:
326	SELFF	739 747:
327	SELFL	737 741:
328	SELFL1	742 745: 745
329	SELFS	740 748:
330	SELFW	738 746:
331	SERR01	73: 75
332	SERR02	74 76:
333	SERROR	71: 195 328 541 580 754 776 804 822
334	SET	1555# 1557:
335	SETQ	1540# 1543: 1951 1954 1960 1963 1973
336	SETQ1	1542: 1549
337	SETQQ	1563# 1565:
338	SFTYP	475- 909- 974- 1009- 1039- 1049-
339	SFVAL	474- 910- 975- 1012- 1035- 1047-
340	SILENCE	103- 115- 139- 148-
341	SINDEX	1903-
342	SINGLE	100: 1933' 1941' 1946'
343	SINGLET	101: 1953' 1962'
344	SPLIST	1031- 1676-
345	SPTYP	1051-
346	SSTACK	102- 122- 420- 429- 433- 528- 735- 743- 751- 767- 797- 815- 1188- 1212- 1272-
347	STACK	172- 241- 394- 407- 437- 508- 622- 731- 913- 954- 1171- 1196- 1260-
348	STATUS	40 1980

349	STEPEVAL	208'	216#	218:								
350	STOP	67-	102'	122'	1868#	1870:	1871-	1934'	1941'	1946'	1953'	1962'
351	SUB	224-	1377-	1738-	1742-	1747-						
352	SUB1	1727#	1738:									
353	SUBFZ	290-										
354	SUBST	1484#	1486:	1491	1493							
355	SUBST*	1499#	1501:									
356	SUBST1	1486	1489:									
357	SUBST1*	1501	1503:									
358	SUBST2*	1506:	1517	1518								
359	SUBST3*	1511	1514:									
360	SYNONYM	1044#	1046:									
361	T	210'	225'	1262'	1275'	1936	1936	1946	1951	1951	1951	1951
		1965	1965	1965	1969'	1969'	1969'					
362	TAPPLY	560	565:									
363	TATOM	1654-										
364	TERPRI	1834#	1849:	1854-								
365	TEVAL1	184	185:									
366	TEVAL2	252	254:									
367	TEVAL3	261	270:									
368	TEVEX	410	412:									
369	TEVEXL	412	417:									
370	TEVEXW	413	422:									
371	TIMES	1732#	1749:									
372	TLIST	89-	386-	517-	634-	844-	1170-	1175-	1195-	1200-	1236-	1270-
		1337-	1350-	1365-	1388-	1433-	1452-	1549-	1580-	1597-	1630-	1635-
		1662-	1672-	1706-	1717-	1841-	1904-					
373	TNIL	162-	166-	387-	635-	842-	1007-	1029-	1030-	1034-	1038-	1066-
		1087-	1107-	1130-	1154-	1187-	1211-	1227-	1463-	1511-	1656-	1671-
374	TNUMB	585-	891-	1233-								
375	TOPLEVEL	129'	136#	138:								
376	TOPST	111-	221-	228-	373-	529-	768-	1120-	1122-	1127-	1129-	1183-
		1207-										
377	TRUE	1103	1227	1230	1236	1240	1248	1251	1275:			
378	TSELF	736	737:									
379	TST	83	84	86	258	306	312	336	343	345	380	384
		408	410	418	420	451	463	474	475	512	556	594
		601	627	636	682	696	702	719	721	736	745	811
		834	851	862	863	864	887	888	906	907	940	941
		942	1064	1070	1082	1084	1137	1154	1166	1168	1191	1193
		1264	1266	1349	1417	1455	1468	1476	1479	1490	1494	1516
		1519	1543	1545	1615	1836	1885					
380	TUNBD1	451	453:									
381	UDFA	565	577:									
382	UDFA1	578:	585									
383	UDFE	270	317:									
384	UDFER	324	326:									
385	UNBDF	455	480:									
386	UNBDL	453	460:									
387	UNBDL1	463:	465									
388	UNBDL2	462	465:									
389	UNBDS	456	492:									
390	UNBDW	454	472:									
391	UNBIND	173	243	408	448:	772	914	955				
392	UNBINP	450:	531	532	770	798	816					
393	UNTIL	1117#	1126:									
394	UNTIL1	1127:	1130									
395	UNTIL2	1126	1129:									
396	V2M	1932#										
397	VCMC2	1933	1940	1945	1952	1961	1965	1969				
398	WHERE	883#	886:									
399	WHERE1	891	894:									
400	WHERE2	893	897:									

401	WHERE3	901:	921									
402	WHILE	1116#	1119:									
403	WHILE1	1120:	1123									
404	WHILE2	1119	1122:									
405	XCONS	207-	698-	705-	837-	840-	845-	849-	853-	858-	864-	945-
		965-	1399#	1402-	1404:	1406-	1423-	1442-	1445-	1448-	1619-	1628-
406	XMOVE	1906-										
407	XTOPST	337-	344-	346-	383-	389-	396-	402-	482-	632-	683-	720-
		812-	948-	1153-	1181-	1205-	1416-	1478-	1493-	1518-		
408	ZEROP	1757#	1765:									
409	~UNDEF	192'	1240'									
410	~interpreter	53'	1936'	1979'								

APPENDICE G

```
toplevel
? '***** A) les types simples de fonctions
=  ***** A) les types simples de fonctions
Nb d'instructions exécutées : 65
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? (SETQ X 10)
=  10
Nb d'instructions exécutées : 69
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 12

toplevel
? (DE F001 (X) (PRINT 'X '= X))
=  F001
Nb d'instructions exécutées : 61
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? (F001 20)
  X   =   20
=  20
Nb d'instructions exécutées : 170
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 20

toplevel
? X
=  10
Nb d'instructions exécutées : 51
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? (DE F002 X (PRINT 'X '= X))
=  F002
Nb d'instructions exécutées : 61
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? (F002 'A 'B 'C)
  X   =   (A B C)
=  (A B C)
Nb d'instructions exécutées : 218
Nb de CONS réalisés         : 3
Taille maximum de la pile   : 20

toplevel
? (DE F003 (X1 X2 . X3) (PRINT X1 X2 X3))
=  F003
Nb d'instructions exécutées : 61
```

```
  Nb de CONS réalisés          : 0
  Taille maximum de la pile    : 9

  toplevel
  ? (FOO3 1 2 3 4)
    1   2   (3 4)
  =  (3 4)
  Nb d'instructions exécutées : 215
  Nb de CONS réalisés          : 2
  Taille maximum de la pile    : 24

  toplevel
  ? (DE FOO4 (L) (IF (LISTP L) (CONS (CAR L) (FOO4 (CDR L))) (PRSTACK '100)
 NIL))
  =  FOO4
  Nb d'instructions exécutées : 61
  Nb de CONS réalisés          : 0
  Taille maximum de la pile    : 9

  toplevel
  ? (FOO4 '(A B))
                (MOVE TST A1 JUMP (PROGN)) :
                ( NIL )
                UNBIND :
                0
                ( ( L ) ( IF & ...) )
                ( (MOVE A1 A2) : B ...)
                L
                ( B )
                *MARK*
                ( 0 ( & & ) ...)
                (MOVE A1 A2) :
                B
                CONS :
                UNBIND :
                0
                ( ( L ) ( IF & ...) )
                ( (MOVE A1 A2) : A ...)
                L
                ( A B )
                *MARK*
                ( 0 ( & & ) ...)
                (MOVE A1 A2) :
                A
                CONS :
                UNBIND :
                0
                ( ( L ) ( IF & ...) )
                ( (SCVAL A1 'IT) : UNBIND : ...)
                L
                ( 1 2 ...)
                *MARK*
                ( 3 0 ...)
                (SCVAL A1 'IT) :
                UNBIND :
                3
                0
                0
                NIL
                (PRSTAT NIL NIL) :
                ( ( STOP ) )
                *EOS*
  =  (A B)
  Nb d'instructions exécutées : 458
  Nb de CONS réalisés          : 2
  Taille maximum de la pile    : 42
```

```
toplevel
? (DF FFOO (L) (PRINT L))
=  FFOO
Nb d'instructions exécutées : 61
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? (FFOO A (B C) D)
  (A (B C) D)
=  (A (B C) D)
Nb d'instructions exécutées : 116
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 20

toplevel
? (DM MMCONS (L) (IF (NULL (CDDR L)) (CADR L) ['CONS (CADR L) (CONS '
MMCONS (CDDR L))]))
=  MMCONS
Nb d'instructions exécutées : 61
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? (MMCONS 'A 'B 'C)
=  (A B . C)
Nb d'instructions exécutées : 616
Nb de CONS réalisés         : 10
Taille maximum de la pile   : 28

toplevel
? (DM ZEROP (L) (RPLACB L ['EQ (CADR L) 0]))
=  ZEROP
Nb d'instructions exécutées : 61
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? (DE FACT (N) (IF (ZEROP N) 1 (TIMES N (FACT (SUB1 N)))))
=  FACT
Nb d'instructions exécutées : 61
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? (FACT 5)
=  120
Nb d'instructions exécutées : 983
Nb de CONS réalisés         : 3
Taille maximum de la pile   : 77

toplevel
? (FVAL 'FACT)
=  ((N) (IF (EQ N 0) 1 (TIMES N (FACT (SUB1 N)))))
Nb d'instructions exécutées : 79
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 12

toplevel
? (FACT 5)
=  120
Nb d'instructions exécutées : 848
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 77

toplevel
? '***** B) les traitements particuliers
```

```
=  ***** B) les traitements particuliers
Nb d'instructions exécutées : 57
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? (DE NPR1 (L) (IF (NULL L) NIL (PRINT (CAR L)) (NPR1 (CDR L))))
=  NPR1
Nb d'instructions exécutées : 61
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? (NPR1)
=  NIL
Nb d'instructions exécutées : 146
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 19

toplevel
? (NPR1 '(1))
  1
=  NIL
Nb d'instructions exécutées : 290
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 22

toplevel
? (NPR1 '(1 2))
  1
  2
=  NIL
Nb d'instructions exécutées : 428
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 22

toplevel
? (NPR1 '(1 2 3))
  1
  2
  3
=  NIL
Nb d'instructions exécutées : 566
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 22

toplevel
? (NPR1 '(1 2 3 4))
  1
  2
  3
  4
=  NIL
Nb d'instructions exécutées : 704
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 22

toplevel
? (DE CPR1 (L) (IF (NULL L) NIL (PRINT (CAR L)) (CPR2 (CDR L))))
=  CPR1
Nb d'instructions exécutées : 61
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? (DE CPR2 (L) (CPR3 L))
=  CPR2
```

```
Nb d'instructions exécutées : 61
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? (DE CPR3 (L) (CPR1 L))
=  CPR3
Nb d'instructions exécutées : 61
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? (CPR1)
=  NIL
Nb d'instructions exécutées : 146
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 19

toplevel
? (CPR1 '(A))
  A
=  NIL
Nb d'instructions exécutées : 434
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 38

toplevel
? (CPR1 '(A B))
  A
  B
=  NIL
Nb d'instructions exécutées : 680
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 38

toplevel
? (CPR1 '(A B C))
  A
  B
  C
=  NIL
Nb d'instructions exécutées : 926
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 38

toplevel
? (CPR1 '(A B C D))
  A
  B
  C
  D
=  NIL
Nb d'instructions exécutées : 1172
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 38

toplevel
? (ESCAPE FIN (PRIN 1) (PRINT 2))
  1   2
=  2
Nb d'instructions exécutées : 134
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 19

toplevel
? (ESCAPE FIN (PRIN 1) (PRINT 2) (FIN 3) (PRINT 4))
  1   2
```

```
= 3
Nb d'instructions exécutées : 165
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 20

toplevel
? (DE EFO (L) (ESCAPE BID (IF (NULL L) NIL (PRINT (CAR L)) (EFO (CDR L)))))
= EFO
Nb d'instructions exécutées : 61
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? (EFO)
= NIL
Nb d'instructions exécutées : 180
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 25

toplevel
? (EFO '(A))
  A
= NIL
Nb d'instructions exécutées : 364
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 31

toplevel
? (EFO '(A B))
  A
  B
= NIL
Nb d'instructions exécutées : 548
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 37

toplevel
? (EFO '(A B C))
  A
  B
  C
= NIL
Nb d'instructions exécutées : 738
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 43

toplevel
? (EFO '(A B C D))
  A
  B
  C
  D
= NIL
Nb d'instructions exécutées : 934
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 49

toplevel
? (DE SELF1 (L) (IF (NULL L) NIL (PRINT (CAR L)) (SELF (CDR L))))
= SELF1
Nb d'instructions exécutées : 61
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? (SELF1 '(1 2))
  1
```

```
  2
=  NIL
Nb d'instructions exécutées : 450
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 22

toplevel
? (SELF1 '(1 2 3))
  1
  2
  3
=  NIL
Nb d'instructions exécutées : 599
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 22

toplevel
? (DE SELF2 (L) (ESCAPE BID (IF (NULL L) NIL (PRINT (CAR L)) (SELF (CDR L))
)))
=  SELF2
Nb d'instructions exécutées : 61
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? (SELF2 '(1 2))
  1
  2
=  NIL
Nb d'instructions exécutées : 582
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 37

toplevel
? (SELF2 '(1 2 3))
  1
  2
  3
=  NIL
Nb d'instructions exécutées : 795
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 43

toplevel
? '***** C) formes spéciales
=  ***** C) formes spéciales
Nb d'instructions exécutées : 57
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? (SETQ X 10)
=  10
Nb d'instructions exécutées : 69
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 12

toplevel
? ((LAMBDA (X) (ADD1 X)) 20)
=  21
Nb d'instructions exécutées : 123
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 17

toplevel
? X
=  10
```

```
Nb d'instructions exécutées : 51
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? (LET NIL (PRINT X))
  10
=  10
Nb d'instructions exécutées : 139
Nb de CONS réalisés         : 4
Taille maximum de la pile   : 18

toplevel
? (LET (X 20) (PRINT X))
  20
=  20
Nb d'instructions exécutées : 168
Nb de CONS réalisés         : 6
Taille maximum de la pile   : 20

toplevel
? (LET ((X 30) (Y 40)) (PRINT X Y))
  30   40
=  40
Nb d'instructions exécutées : 221
Nb de CONS réalisés         : 8
Taille maximum de la pile   : 22

toplevel
? (SETQ L '(LET (X 2) (PRINT X)))
=  (LET (X 2) (PRINT X))
Nb d'instructions exécutées : 76
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 12

toplevel
? (EVAL L)
  2
=  2
Nb d'instructions exécutées : 203
Nb de CONS réalisés         : 6
Taille maximum de la pile   : 20

toplevel
? L
=  ((LAMBDA (X) (PRINT X)) 2)
Nb d'instructions exécutées : 51
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? ((ADD1 3) '(A B C D E F G))
=  D
Nb d'instructions exécutées : 102
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 11

toplevel
? ((CAR '(CDR)) '(A B C))
=  (B C)
Nb d'instructions exécutées : 98
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 12

toplevel
? ((INTERNAL 7 ((X) (CONS 'ADD1 X))) (CONS (ADD1 19)))
=  (ADD1 20)
```

```
Nb d'instructions exécutées : 175
Nb de CONS réalisés         : 2
Taille maximum de la pile   : 20

toplevel
? X
=  10
Nb d'instructions exécutées : 51
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? ((INTERNAL 8 ((X) (CONS 'A X))) (CONS (ADD1 19)))
=  (A (CONS (ADD1 19)))
Nb d'instructions exécutées : 125
Nb de CONS réalisés         : 1
Taille maximum de la pile   : 20

toplevel
? X
=  10
Nb d'instructions exécutées : 51
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? ((INTERNAL 9 ((X) (CONS 'ADD1 (CADR X)))) ((ADD1 19)))
=  21
Nb d'instructions exécutées : 165
Nb de CONS réalisés         : 1
Taille maximum de la pile   : 21

toplevel
? (WHERE (CAR '((X) (CDR X))) (CAR '(A B C D)))
=  (B C D)
Nb d'instructions exécutées : 203
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 23

toplevel
? (WHERE (CDR 7 '((X) (CAR X))) (CDR '(A B C)))
=  A
Nb d'instructions exécutées : 190
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 23

toplevel
? (APPLY 'CONS '((ADD1 1) (ADD1 2) (ADD1 3)))
=  ((ADD1 1) ADD1 2)
Nb d'instructions exécutées : 93
Nb de CONS réalisés         : 1
Taille maximum de la pile   : 12

toplevel
? (APPLY 'LIST '((ADD1 1) (ADD1 2) (ADD1 3)))
=  (2 3 4)
Nb d'instructions exécutées : 159
Nb de CONS réalisés         : 3
Taille maximum de la pile   : 13

toplevel
? (MAP (LAMBDA (X Y Z) (PRINT X Y Z)) '(A B C) '(D E))
  (A B C)   (D E)   NIL
  (B C)   (E)   NIL
  (C)   NIL   NIL
=  NIL
Nb d'instructions exécutées : 576
```

```
Nb de CONS réalisés              : 11
Taille maximum de la pile        : 29

toplevel
? (MAP (LAMBDA (X . Y) (PRINT X Y)) '(A B C) '(D E F) '(G H I J) '(K))
  (A B C)    ((D E F) (G H I J) (K))
  (B C)    ((E F) (H I J) NIL)
  (C)    ((F) (I J) NIL)
  NIL    (NIL (J) NIL)
=  NIL
Nb d'instructions exécutées : 713
Nb de CONS réalisés         : 25
Taille maximum de la pile   : 29

toplevel
? (MAPC (LAMBDA (X Y Z T) (PRINT X Y Z T)) 'A '(B . C) '(D E) '(F G H))
  A    B    D    F
  A    C    E    G
  A    C    NIL    H
=  NIL
Nb d'instructions exécutées : 748
Nb de CONS réalisés         : 21
Taille maximum de la pile   : 33

toplevel
? '***** D) les variable-fonctions
=  ***** D) les variable-fonctions
Nb d'instructions exécutées : 57
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? (DE VAR (N) (IF N (SETQ VAR N) VAR))
=  VAR
Nb d'instructions exécutées : 61
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? (VAR 10)
=  10
Nb d'instructions exécutées : 152
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 20

toplevel
? (LETF (VAR 11) (PRINT (VAR)))
  11
=  11
Nb d'instructions exécutées : 476
Nb de CONS réalisés         : 1
Taille maximum de la pile   : 27

toplevel
? (VAR)
=  10
Nb d'instructions exécutées : 137
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 19

toplevel
? (VAR '(X Y Z))
=  (X Y Z)
Nb d'instructions exécutées : 159
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 20
```

```
toplevel
? (LETF (VAR '(A B)) (VAR))
=  (A B)
Nb d'instructions exécutées : 462
Nb de CONS réalisés         : 1
Taille maximum de la pile   : 24

toplevel
? (PROGN (PRINT (VAR)) (ESCAPE ECH (LETF (VAR '(1 2)) (PRINT (VAR)) (ECH)))
(VAR))
   (X Y Z)
   (1 2)
=  (X Y Z)
Nb d'instructions exécutées : 778
Nb de CONS réalisés         : 1
Taille maximum de la pile   : 37

toplevel
? '----- 1 - La fonction de FIBONACCI
=  ----- 1 - La fonction de FIBONACCI
Nb d'instructions exécutées : 57
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? (DE FIB (N) (COND ((ZEROP N) 1) ((EQ N 1) 1) (T (PLUS (FIB (SUB1 N)) (
FIB (DIFFER N 2))))))
=  FIB
Nb d'instructions exécutées : 61
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? (FIB 8)
=  34
Nb d'instructions exécutées : 11636
Nb de CONS réalisés         : 3
Taille maximum de la pile   : 100

toplevel
? '----- 2 - La fonction d'ACKERMANN
=  ----- 2 - La fonction d'ACKERMANN
Nb d'instructions exécutées : 57
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? (DE ACK (X Y) (COND ((ZEROP X) (ADD1 Y)) ((ZEROP Y) (ACK (SUB1 X) 1)) (T
(ACK (SUB1 X) (ACK X (SUB1 Y))))))
=  ACK
Nb d'instructions exécutées : 61
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? (ACK 2 2)
=  7
Nb d'instructions exécutées : 4749
Nb de CONS réalisés         : 6
Taille maximum de la pile   : 111

toplevel
? '----- 3 - Un bien étrange REVERSE
=  ----- 3 - Un bien étrange REVERSE
Nb d'instructions exécutées : 57
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9
```

```
toplevel
? (DE REV (L) (IF (NULL (CDR L)) L (CONS (CAR (REV (CDR L))) (REV (CONS (
CAR L) (REV (CDR (REV (CDR L)))))))))
=  REV
Nb d'instructions exécutées : 61
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? (REV '(A B C D E))
=  (E D C B A)
Nb d'instructions exécutées : 24791
Nb de CONS réalisés         : 94
Taille maximum de la pile   : 120

toplevel
? '----- 4 - Générateur de permutations (co-post-rec)
=  ----- 4 - Générateur de permutations (co-post-rec)
Nb d'instructions exécutées : 57
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? (DE PERMS (N) (F 0 NIL))
=  PERMS
Nb d'instructions exécutées : 61
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? (DE F (NIV E) (IF (LT NIV N) (H (ADD1 NIV) (CONS 1 E)) (PRINT (REVERSE E)
) (G NIV (CAR E) (CDR E))))
=  F
Nb d'instructions exécutées : 61
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? (DE G (NIV X E) (IF (EQ X N) (IF E (G (SUB1 NIV) (CAR E) (CDR E)) 'FINI)
(H NIV (CONS (ADD1 X) E))))
=  G
Nb d'instructions exécutées : 61
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? (DE H (NIV E) (IF (MEMQ (CAR E) (CDR E)) (G NIV (CAR E) (CDR E)) (F NIV
E)))
=  H
Nb d'instructions exécutées : 61
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? (PERMS 1)
  (1)
=  FINI
Nb d'instructions exécutées : 821
Nb de CONS réalisés         : 2
Taille maximum de la pile   : 54

toplevel
? (PERMS 2)
  (1 2)
  (2 1)
=  FINI
```

```
Nb d'instructions exécutées : 2926
Nb de CONS réalisés         : 10
Taille maximum de la pile   : 59

toplevel
? (PERMS 3)
   (1 2 3)
   (1 3 2)
   (2 1 3)
   (2 3 1)
   (3 1 2)
   (3 2 1)
=  FINI
Nb d'instructions exécutées : 12576
Nb de CONS réalisés         : 48
Taille maximum de la pile   : 59

toplevel
? '***** E)  Test des erreurs
= ***** E)  Test des erreurs
Nb d'instructions exécutées : 57
Nb de CONS réalisés         : 0
Taille maximum de la pile   : 9

toplevel
? FOOBARX25

***   EVAL : Variable indéfinie.
Dernière forme évaluée :  FOOBARX25
            UNBIND :
            3
            ( 3 0 ...)
            1
            NIL
            (SCVAL A1 'IT) :
            UNBIND :
            3
            0
            0
            NIL
            (PRSTAT NIL NIL) :
            ( ( STOP ) )
            *EOS*
=  NIL
Nb d'instructions exécutées : 114
Nb de CONS réalisés         : 2
Taille maximum de la pile   : 17

toplevel
? (UDFE '(A))

***   EVAL : fonction indéfinie :  UDFE
Dernière forme évaluée :  (UDFE '(A))
            UNBIND :
            3
            ( 3 0 ...)
            1
            NIL
            (SCVAL A1 'IT) :
            UNBIND :
            3
            0
            0
            NIL
            (PRSTAT NIL NIL) :
            ( ( STOP ) )
            *EOS*
```

```
= NIL
Nb d'instructions exécutées : 131
Nb de CONS réalisés         : 3
Taille maximum de la pile   : 17

toplevel
? (SELF 1)

***  Erreur SELF :  (1)
Dernière forme évaluée :  (SELF 1)
            UNBIND :
            3
            ( 3 0 ...)
            1
            NIL
            (SCVAL A1 'IT) :
            UNBIND :
            3
            0
            0
            NIL
            (PRSTAT NIL NIL) :
            ( ( STOP ) )
            *EOS*
= NIL
Nb d'instructions exécutées : 138
Nb de CONS réalisés         : 3
Taille maximum de la pile   : 17

toplevel
? (EXIT 2)

***  Erreur EXIT :  (2)
Dernière forme évaluée :  (EXIT 2)
            UNBIND :
            3
            0
            0
            NIL
            (PRSTAT NIL NIL) :
            ( ( STOP ) )
            *EOS*
Nb d'instructions exécutées : 126
Nb de CONS réalisés         : 3
Taille maximum de la pile   : 11

toplevel
? (APPLY 'UDFA 1)

***  APPLY : fonction indéfinie :  UDFA
Dernière forme évaluée :  1
            UNBIND :
            3
            ( 3 0 ...)
            1
            NIL
            (SCVAL A1 'IT) :
            UNBIND :
            3
            0
            0
            NIL
            (PRSTAT NIL NIL) :
            ( ( STOP ) )
            *EOS*
= NIL
Nb d'instructions exécutées : 155
```

```
Nb de CONS réalisés          : 3
Taille maximum de la pile    : 17

toplevel
? (WHERE (SUPESCAPE 10) (SUPESCAPE 'OK))
=  OK
Nb d'instructions exécutées : 126
Nb de CONS réalisés          : 0
Taille maximum de la pile    : 18

toplevel
? (STOP)
Bye
Nb d'instructions exécutées : 81775
```